UNIVERSITÉ DE SHERBROOKE
Faculté d'éducation

Conception d'une grille d'analyse des langages de programmation utilisés en algorithmique dans le programme Techniques de l'informatique

Par
Stéphane Duguay

Essai présenté à la Faculté d'éducation
en vue de l'obtention du grade de
Maître en enseignement (M.Éd.)
Maîtrise en enseignement au collégial

Mai 2018

UNIVERSITÉ DE SHERBROOKE
Faculté d'éducation

Conception d'une grille d'analyse des langages de programmation utilisés en algorithmique dans le programme Techniques de l'informatique

Par
Stéphane Duguay

a été évalué par un jury composé des personnes suivantes :

Christian Barrette	Directeur d'essai
Christian Potvin	Évaluateur de l'essai

Essai accepté le :

REMERCIEMENTS

Je remercie chaleureusement monsieur Christian Barrette, le directeur de cet essai, qui a su m'accompagner avec pertinence et patience lors des différentes étapes de recherche et de rédaction.

Je tiens aussi à remercier madame Denyse Lemay, monsieur Jacques Lecavalier et madame Sawsen Lakhal pour leur guidance lors des cours MEC803, MEC801 et MEC800 ainsi que madame Annie-Claude Prud'homme, répondante PERFORMA au Cégep de Rimouski.

Finalement, je remercie ma conjointe et mes enfants pour leur soutien indéfectible.

SOMMAIRE

L'algorithmique est le fondement logique de la programmation. La sélection d'un langage de programmation pour l'apprentissage et l'enseignement de l'algorithmique pour ordinateurs est une décision d'une importance capitale. En effet, les répercussions de ce choix sont multiples et affectent profondément les étudiants et étudiantes. La personne ou l'équipe responsable de ce choix porte une lourde responsabilité lors de cette prise de décision. Malheureusement, peu d'outils encadrent la réflexion sur ce sujet. Cet essai décrit le processus de conception d'une grille d'analyse des langages de programmation permettant de guider le, la ou les responsables de ce choix. Concrètement, l'objectif général de cette recherche développement est de concevoir une grille d'analyse des langages de programmation pour l'apprentissage et l'enseignement de l'algorithmique dans le cadre du programme Techniques de l'informatique au collégial québécois pour permettre à un enseignant, une enseignante ou une équipe de sélectionner le langage de programmation le plus approprié pour leurs étudiants et étudiantes, et pour faciliter leur rapport aux savoirs algorithmiques.

À l'aide d'un cadre de référence de questionnement didactique selon Lapierre (2008) et en y intégrant divers cadres de références secondaires, cette recherche procédera à l'atteinte des objectifs spécifiques suivants :

1. Dégager les attributs d'un langage de programmation pour l'apprentissage et l'enseignement de l'algorithmique adapté aux besoins des cours de programmation introductifs en Techniques de l'informatique au niveau collégial québécois.
2. Mettre au point une grille d'analyse des langages de programmation à partir des attributs dégagés.
3. Valider la grille d'analyse auprès de pairs quant à son utilité pour le choix d'un langage de programmation approprié.

La méthodologie encadrant la démarche de recherche est basée sur le modèle de recherche développement proposé par Loiselle et Harvey (2009). Ce modèle propose cinq phases macroscopiques : l'origine de la recherche, le référentiel, la méthodologie, l'opérationnalisation et les résultats. La phase « origine de la recherche » du modèle est détaillée dans le chapitre 1 de cet essai, la problématique, tandis que la phase « référentiel » est décrite dans le chapitre 2, le cadre de référence. Le troisième chapitre de cet essai décrit la phase « méthodologie » de la recherche développement. Finalement, les phases « opérationnalisation » et « résultats » du modèle de Loiselle et Harvey sont présentés ensemble dans le dernier chapitre de l'essai : la présentation et l'interprétation des résultats.

Une première version de la grille d'analyse des langages de programmation est produite pour être ensuite soumise à des pairs pour une collecte de données. L'analyse de ces données permettra la création d'une version améliorée de la grille : ce sera le produit final issu du processus de développement documenté dans cet essai et elle est présentée à la fin du chapitre 4.

TABLE DES MATIÈRES

REMERCIEMENTS **3**

SOMMAIRE **5**

LISTE DES TABLEAUX **11**

LISTE DES FIGURES **13**

INTRODUCTION **15**

PREMIER CHAPITRE LA PROBLÉMATIQUE **17**

1. LE CONTEXTE DE LA RECHERCHE 17

1.1 De l'algorithmique… 17

1.2 … à la programmation 20

1.3 L'algorithmique et la profession de programmeur-analyste 21

1.4 L'algorithmique au collégial québécois 22

2. LE PROBLÈME DE RECHERCHE 24

2.1 Les effets du choix de langage de programmation sur l'apprentissage et l'enseignement 25

2.2 Le choix d'un langage pour débuter 25

3. L'OBJECTIF GÉNÉRAL DE RECHERCHE 27

DEUXIÈME CHAPITRE LE CADRE DE RÉFÉRENCE **29**

1. LE QUESTIONNEMENT DIDACTIQUE 30

1.1 Les savoirs disciplinaires et professionnels 31

1.2 Les savoirs à enseigner 34

1.3 Le rapport des étudiants aux savoirs 35

1.4 Le matériel didactique 37

1.5 Les stratégies d'enseignement, d'apprentissage et d'évaluation 41

2. SYNTHÈSE DU CADRE DE RÉFÉRENCE : DES OPTIQUES À CONSIDÉRER LORS DE LA CRÉATION DE LA GRILLE D'ANALYSE.....42

3. LES OBJECTIFS SPÉCIFIQUES 46

TROISIÈME CHAPITRE LA MÉTHODOLOGIE.....47

1. LA RECHERCHE DÉVELOPPEMENT 47

2. L'APPROCHE MÉTHODOLOGIQUE.....48

3. LE DÉROULEMENT DE LA RECHERCHE.....49

3.1 Calendrier de la recherche.....51

4. LA COLLECTE DES DONNÉES.....52

4.1 Les participants et participantes à la recherche.....52

4.2 Le journal de bord 55

4.3 La mise à l'essai et l'entrevue semi-dirigée.....55

5. L'ANALYSE DES DONNÉES.....57

5.1 Le journal de bord 58

5.2 Les entrevues.....58

6. MOYENS POUR ASSURER LA RIGUEUR ET LA SCIENTIFICITÉ.....59

6.1 La rigueur en recherche qualitative.....59

6.2 La scientificité en recherche développement 62

7. ASPECTS ÉTHIQUES DE LA RECHERCHE.....63

QUATRIÈME CHAPITRE LA PRÉSENTATION ET L'INTERPRÉTATION DES RÉSULTATS.....65

1. LA DÉRIVATION DES ATTRIBUTS 65

2. LA VERSION INITIALE DE LA GRILLE D'ANALYSE.....77

3. LES DONNÉES COLLECTÉES.....90

Personne participante nº1 90

Personne participante nº2 90

Personne participante nº3 91

Personne participante nº4 92

4. L'ANALYSE DES DONNÉES COLLECTÉES 93

5. LA VERSION AMÉLIORÉE DE LA GRILLE 97

CONCLUSION 111

RÉFÉRENCES BIBLIOGRAPHIQUES 113

ANNEXE A LA COMPETENCE 016W DU DEVIS MINISTERIEL POUR TECHNIQUES DE L'INFORMATIQUE (GOUVERNEMENT DU QUEBEC, 2000) : « PRODUIRE DES ALGORITHMES » 119

ANNEXE B LA COMPETENCE 016S DU DEVIS MINISTERIEL POUR TECHNIQUES DE L'INFORMATIQUE (GOUVERNEMENT DU QUEBEC, 2000) : « EXPLOITER UN LANGAGE DE PROGRAMMATION STRUCTUREE » 123

ANNEXE C PLAN POUR L'ENTREVUE SEMI-DIRIGEE 127

ANNEXE D GRILLE DE CLASSIFICATION DES REPONSES COLLECTEES 131

ANNEXE E LETTRE D'INFORMATION ET FORMULAIRE DE CONSENTEMENT 135

LISTE DES TABLEAUX

Tableau 1 Exemple d'algorithme en pseudocode 23
Tableau 2 Résultat de l'exécution manuelle d'un algorithme 24
Tableau 3 Les optiques à considérer 43
Tableau 4 Calendrier de la recherche 51
Tableau 5 Stratégies de dérivation des attributs 67
Tableau 6 Version initiale de la grille d'analyse des langages de programmation 78
Tableau 7 Version améliorée de la grille d'analyse des langages de programmation 98
Tableau 8 Énoncé de la compétence « Produire des algorithmes » 121
Tableau 9 Énoncé de la compétence « Exploiter un langage de programmation structurée » 125
Tableau 10 Grille de classification des réponses collectées 133

LISTE DES FIGURES

Figure 1 Modèle de questionnement didactique (Prud'homme, 2015, p. 40)..... 30
Figure 2 Modèle de recherche développement en éducation (Loiselle et Harvey, 2009, p. 110) 50

INTRODUCTION

Cet essai traite de la conception d'une grille d'analyse des langages de programmation pour l'apprentissage et l'enseignement de l'algorithmique dans le cadre du programme de Techniques de l'informatique au collégial québécois. Il s'intéresse principalement aux attributs des langages de programmation dans un contexte éducatif.

Le contexte éducatif qui sera abordé dans cet essai est un cours d'introduction à l'algorithmique dans lequel les étudiants et étudiantes utilisent un langage de programmation pour l'écriture d'algorithmes permettant la résolution d'exercices pratiques, mais aussi pour la lecture d'algorithmes exemples accompagnant des concepts théoriques.

En travaillant avec un langage de programmation, les étudiants et étudiantes doivent en apprendre le fonctionnement de base en plus de développer leur maitrise de l'algorithmique. Si le langage choisi pour apprendre la programmation est complexe, il monopolise les efforts de compréhension et le temps de l'étudiant ou l'étudiante, au détriment de l'apprentissage de l'algorithmique, objet essentiel du cours. Il ou elle peut se retrouver dans la situation d'échouer sur les deux fronts : ne pas apprendre véritablement l'algorithmique et ne pas maitriser le langage de programmation pour exprimer l'algorithmique.

Le choix du langage de programmation est donc une décision didactique fondamentale lors de la mise au point d'un cours d'introduction à l'algorithmique. Dans le but d'assister la ou les personnes responsables de ce choix primordial, nous proposons une grille d'analyse des langages de programmation conçue à l'aide de l'application exhaustive d'un cadre de référence de questionnement didactique. Cette analyse didactique permettra de dégager les considérations en lien avec les savoirs disciplinaires et professionnels, les savoirs à enseigner, les rapports des étudiants et

étudiantes aux savoirs, le matériel didactique et les stratégies d'enseignement, d'apprentissage et d'évaluation.

Cet essai détaille le processus de conception et de validation déployé à l'aide d'une démarche méthodologique de recherche développement. Ce type de recherche mène à la création d'un produit concret, utile et diffusable, qui permettra d'assister la ou les personnes responsables de choisir le langage de programmation à utiliser.

Nous introduirons premièrement la problématique pour ensuite déployer le cadre de référence. L'approche méthodologique suivra pour terminer finalement par la présentation et l'interprétation des résultats.

PREMIER CHAPITRE
LA PROBLÉMATIQUE

Ce chapitre introduit le contexte dans lequel s'inscrit la conception de la grille d'analyse des langages de programmation pour l'apprentissage et l'enseignement de l'algorithmique pour ensuite s'intéresser spécifiquement au problème de recherche en terminant par l'objectif général de recherche.

1. LE CONTEXTE DE LA RECHERCHE

Cette section décrira de façon générale la discipline de l'algorithmique pour ensuite faire le lien entre celle-ci et la programmation. Suivra une exploration de l'importance de l'algorithmique sur le marché du travail justifiant la place qu'elle prend dans le programme de formation Techniques de l'informatique au collégial québécois.

1.1 De l'algorithmique…

L'algorithmique est la discipline couvrant « la connaissance des techniques utilisées pour construire des algorithmes et les méthodes d'investigation permettant de créer des algorithmes répondant à un problème donné » (Office québécois de la langue française, s.d.).

Dans sa définition la plus simple, un algorithme est « un ensemble d'étapes qui permettent d'accomplir une tâche » (Cormen, 2013, p. 1). Par exemple, pour la classe de problèmes « Est-ce que N est un nombre pair? », pour n'importe quelle valeur numérique entière de N, on peut créer un algorithme permettant à une personne ou un ordinateur de répondre OUI ou NON.

Les algorithmes existent depuis très longtemps, avant même que l'idée des ordinateurs existe. Un des premiers algorithmes élaborés est l'anthyphérèse d'Euclide qui permet de déterminer le plus grand diviseur commun entre deux nombres naturels

non nuls ; nous l'avons tous appris lors de notre passage à l'école secondaire. En fait, nous sommes entourés d'algorithmes s'adressant aux humains. Par exemple, une recette de gâteau au chocolat est un algorithme permettant de résoudre le problème « Comment faire un gâteau au chocolat ? » Il en va de même pour le livret contenant les instructions d'assemblage accompagnant un meuble à assembler : il contient l'algorithme permettant de résoudre le problème « Comment assembler les pièces séparées en un meuble ? »

Une des difficultés de l'algorithmique est que, pour un problème complexe, le nombre d'étapes à expliciter peut être très grand. Ce nombre considérable d'étapes empêche l'écriture spontanée d'un algorithme.

Puisque toutes les étapes d'un algorithme ne peuvent pas être écrites d'un seul coup, on doit plutôt décomposer le problème complexe initial en sous-problèmes plus simples. Chaque sous-problème est ensuite analysé et une évaluation de sa complexité est faite. Si ce sous-problème est « trop grand » (chacun juge de cela par lui-même d'après ses capacités ou contraintes), on pourra décomposer ce sous-problème en des sous-sous-problèmes et ainsi de suite jusqu'à ce qu'on ait plusieurs petits problèmes de taille « acceptable » à régler au lieu d'un seul problème d'une grande complexité. Cette approche de détermination d'une séquence logique des opérations est ce qu'on peut appeler la décomposition successive des opérations.

Prenons un autre exemple de la vie courante : comment faire une tasse de thé. Imaginons que nous tentions de donner ces instructions à un robot. Voilà comment on pourrait lui décomposer le travail dans un premier temps :

```
Faire une tasse de thé :
     Obtenir un sachet de thé
     Faire bouillir de l'eau
     Prendre une tasse
```

```
    Mettre le sachet de thé dans la tasse
    Verser de l'eau bouillante dans la tasse
    Attendre 5 minutes
    Retirer le sachet
```

Maintenant, dépendant des fonctionnalités disponibles sur le robot, on pourrait avoir à lui expliciter l'étape « Prendre une tasse » :

```
Prendre une tasse :
    Se déplacer vers l'armoire à tasses
    Ouvrir la porte de l'armoire à tasses
    Tendre le bras vers une tasse
    Refermer les doigts autour de l'anse de la tasse
    Rétracter le bras
    Refermer la porte de l'armoire à tasses
    Retourner à l'endroit initial
```

On peut décomposer aussi finement que nécessaire chaque opération en sous-opérations. Cette tâche est plutôt abstraite et demande un travail cognitif important. En effet, la décomposition successive des opérations demande de « décomposer une situation problématique donnée en ses parties constituantes et, subséquemment, d'identifier les rapports qui existent entre ces parties et d'en dévoiler les principes d'organisation sous-jacents c'est-à-dire de résoudre le problème » (Prégent, 1990, p. 39-40).

Dans le cadre de l'informatique, l'algorithmique est appliquée à la programmation des ordinateurs. On veut pouvoir définir les étapes nécessaires à la résolution d'un problème pour que l'ordinateur puisse exécuter ces étapes et nous fournir le résultat. Tous les programmes informatiques sont des constructions

d’algorithmes. La maitrise de la pensée algorithmique est un prérequis pour l’apprentissage de la programmation.

1.2 … à la programmation

Lorsqu’il est destiné à un ordinateur, un algorithme est « un ensemble d’étapes qui permettent d’accomplir une tâche et qui est décrit de manière suffisamment précise pour qu’un ordinateur puisse l’exécuter » (Cormen, 2013, p.2).

Pour la construction d’un programme informatique, les opérations d’un algorithme sont décomposées jusqu’à ce qu’on puisse les arrimer aux instructions élémentaires d’un langage permettant de codifier des algorithmes pour ordinateurs. C’est ce qu’on appelle la programmation. Comme dans une recette de cuisine, c’est l’ensemble des instructions qui forme la procédure à suivre. Dans un ordinateur, il existe une multitude d’instructions possibles. De plus, chaque langage de programmation permet des instructions différentes tout en utilisant une syntaxe qui lui est propre.

En plus des instructions numériques, booléennes et de certaines permettant la manipulation de données textuelles, la plupart des langages fournissent aussi des instructions de sortie pour l’écriture à l’écran, à l’imprimante ou encore dans des fichiers de sortie. On peut habituellement aussi émettre du son et des commandes graphiques permettant de dessiner sur un de ces trois supports. À l’inverse, on peut obtenir des données, des entrées, à partir du clavier, de fichiers, mais aussi de différents périphériques tels que souris, caméra, micro, scanneur ou écran tactile.

Les instructions d’un programme sont donc les transpositions, dans un langage de programmation, des opérations algorithmiques qui furent décomposées successivement. Ces transpositions ne sont pas toujours directes et l'on doit parfois organiser les instructions entre elles à l’aide de structures de contrôle : ce sont les

modes d'organisation des instructions. Il n'existe que trois structures de contrôle dans un ordinateur : la séquence, l'alternative (aussi appelée *conditionnelle*) et la répétitive (aussi appelée *boucle*).

Si les rudiments de la discipline d'algorithmique s'enseignent et s'apprennent relativement facilement, l'apprenant, pour devenir efficace, doit acquérir de l'expérience dans différents contextes et développer une certaine intuition. Dans ce sens, l'algorithmique constitue une discipline au seuil bas, mais au plafond élevé, c'est-à-dire que ses rudiments simples conduisent rapidement à la réalisation de tâches complexes sollicitant des opérations cognitives de haut niveau.

1.3 L'algorithmique et la profession de programmeur-analyste

La capacité à mettre au point des algorithmes est nécessaire à une carrière en programmation puisque, comme démontré précédemment, cette activité est un dérivé de l'algorithmique. À la sortie de leur formation collégiale, « les programmeurs-analystes/programmeuses-analystes commencent généralement par assurer le développement du code uniquement » (Gouvernement du Québec, 2013, p. 69). Le code est l'ensemble des instructions du programme. Le programmeur ou la programmeuse a la responsabilité d'« écrire, modifier, intégrer et mettre à l'essai le code informatique pour des applications logicielles » (*Ibid.*, p. 47). Concrètement, « le résultat attendu […] est un jeu vidéo, un programme, une application ou un logiciel qui ne contienne pas d'erreurs ou de failles dans le code » (*Ibid.*, p. 69).

Pour réaliser cette tâche, il ou elle doit avoir la « capacité d'interpréter, de choisir et d'adapter des algorithmes. Ces connaissances sont à la base de la résolution des problèmes en informatique » (Gouvernement du Québec, 2015, p. 39).

Au niveau cognitif, « la capacité de résolution de problèmes est au cœur de la profession et se manifeste en grande partie au moment de l'utilisation des algorithmes et de la programmation » (*Ibid.*, p. 42). Notons au passage que :

> Les techniciennes et les techniciens en informatique utilisent un grand nombre de langages et de techniques de programmation qui peuvent être groupés dans les catégories (qui ne sont pas nécessairement exclusives) ou les paradigmes de programmation suivants : structurés […]; orientés objets […]; fonctionnels […]. (*Ibid.*, p. 39).

1.4 L'algorithmique au collégial québécois

La capacité de l'apprenant ou de l'apprenante à mettre au point des algorithmes est aussi utile tout au long de son programme de formation. En effet, l'algorithmique est au cœur du premier cours de programmation offert en Techniques de l'informatique au Cégep de Rimouski. Il s'agit d'un des piliers du programme de formation : tous les autres cours de programmation du programme de formation ont ce cours comme prérequis, soit directement ou indirectement. Il est à l'horaire au premier trimestre et c'est un cours de six heures de classe par semaine. En moyenne, trois heures par semaine sont consacrées aux explications magistrales et les trois autres sont assignées aux activités d'apprentissage. De plus, on exige des apprenants et apprenantes, en moyenne, trois heures de travail à la maison.

Dans le devis ministériel du programme de Techniques de l'informatique (Gouvernement du Québec, 2000), la compétence « 016W : Produire des algorithmes » couvre explicitement l'algorithmique, séparément de la programmation. C'est la seule compétence du devis qui mentionne l'algorithmique.

Le verbe « Produire » utilisé pour décrire la compétence implique la maitrise complète du processus de création des algorithmes. Cette activité est de haut niveau cognitif puisqu'elle se situe au niveau le plus complexe de la taxonomie du domaine cognitif révisée d'Anderson et Krathwohl (Krathwohl, 2002) : la création à l'aide de connaissances factuelles, conceptuelles, procédurales et métacognitives.

L'élément de compétence « 016W.2 : Mettre au point l'algorithme » est central à la maitrise de la pensée algorithmique et plus précisément ses critères de performance « 016W.2.2 : Détermination d'une séquence logique des opérations » et « 016W.2.3 : Détermination des structures de traitement appropriées à chacune des opérations. ». L'annexe A de cet essai présente plus de détails sur la compétence 016W.

Dans la classe, l'apprentissage et l'enseignement des concepts de base de l'algorithmique peuvent se dérouler sur papier, sans avoir besoin d'un ordinateur. La conception manuscrite peut se faire à l'aide du pseudocode : un langage permettant de décrire les opérations algorithmiques. Le pseudocode est un langage souple, sans syntaxe accablante, qui permet d'écrire des algorithmes dans n'importe quelle langue humaine. Dans le réseau d'enseignement collégial québécois francophone, nous écrivons habituellement les algorithmes en français (`si`, `alors`, `sinon`, `tant que`, etc.). Voici un exemple d'algorithme permettant d'écrire à l'écran les nombres de 1 à 5 avec des commentaires pour expliquer chaque opération.

Tableau 1 Exemple d'algorithme en pseudocode

Pseudocode	**Commentaires**
`X ← 1`	Assigner 1 à la variable X
`Tant que X <= 5`	Répétitive tant que X est plus petit ou égal à 5
`    Écrire X`	Écrire sur une ligne la valeur de X à l'écran
`    X ← X + 1`	Incrémenter X de 1

On peut faire exécuter cet algorithme sur papier par l'apprenant ou l'apprenante en lui faisant jouer le rôle de l'ordinateur, c'est ce qu'on appelle l'exécution manuelle. Voici ce que donnerait l'exécution manuelle de l'algorithme ci-dessus :

Tableau 2 Résultat de l'exécution manuelle d'un algorithme

Mémoire	**Écran**
Valeurs de X : ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6	1 2 3 4 5

Malgré ses acquis initiaux réalisés lors de cet apprentissage en mode manuel, l'apprenant ou apprenante doit relativement rapidement les transférer sur ordinateur. Il ou elle doit programmer ses algorithmes parce que les concepts de base peuvent s'acquérir en quelques rencontres seulement et qu'on voudra permettre la création d'algorithmes plus complexes que ceux qui sont écrits et exécutés sur papier. Cependant, programmer sur l'ordinateur implique l'introduction d'une syntaxe formelle, celle du langage de programmation sélectionné. Cela implique que l'on doit maintenant s'occuper des dimensions plus techniques de la solution. La complexité a augmenté, car en plus de mettre en pratique l'algorithmique, on doit prendre en considération la syntaxe du langage et de ses instructions.

2. LE PROBLÈME DE RECHERCHE

Le choix d'un langage de programmation permettant de mettre en pratique des applications de plus en plus complexes de l'algorithmique est un choix didactique primordial, car ses effets sont multiples, autant sur les étudiants et étudiantes et leur rapport aux savoirs que sur l'enseignant ou l'enseignante et les stratégies d'enseignement et d'évaluation déployées. La personne ou l'équipe responsable de prendre cette décision ont plusieurs aspects à prendre en considération.

2.1 Les effets du choix de langage de programmation sur l'apprentissage et l'enseignement

Lorsque des concepts d'algorithmique plus avancés que ceux se manipulant sur papier sont enseignés, l'enseignant ou l'enseignante doit aussi enseigner les considérations techniques et procédurales de leurs solutions. En travaillant avec un langage de programmation, surtout s'il est destiné aux professionnels, les apprenants et apprenantes doivent nécessairement en apprendre le fonctionnement de base en plus d'en perfectionner leur compréhension en cours de route. Il y a interférence avec le but principal du cours qui est d'acquérir la compétence de production d'algorithmes, et les étudiants ou étudiantes qui ne peuvent gérer cela échouent (Yadin, 2013, p. 15-16).

L'apprentissage des détails techniques entourant l'utilisation d'un langage et d'un environnement de développement requiert du temps. L'enseignant ou l'enseignante se retrouve donc devant la nécessité de choisir entre accorder du temps à l'apprentissage et l'enseignement du langage et de l'environnement ou accorder du temps à l'apprentissage et l'enseignement de l'algorithmique et à son application à différents problèmes concrets.

Ces problèmes ont des impacts profonds sur la réussite puisque « le taux d'échec ou d'abandon aux cours d'initiation à la programmation en premier cycle universitaire varient de 25 à 80% de par le monde. » (Benabbou et Hanoune, 2006, p. 3). Ces échecs ne sont pas occasionnés nécessairement par la mauvaise compréhension des concepts algorithmiques présentés dans le cadre du cours, mais par un langage de programmation inapproprié (Yadin, 2011, p. 73).

2.2 Le choix d'un langage pour débuter

Il existe des environnements et des langages pour l'apprentissage de la programmation. Ces langages sont spécifiquement conçus pour permettre de réduire la complexité habituellement associée aux langages de programmation professionnels et

donc de réduire l'interférence qu'ils ont sur l'application des concepts algorithmiques. Par exemple, certaines universités américaines proposent un langage et le rendent disponible pour tous : mentionnons l'environnement Scratch, par le très réputé *Massachusetts Institute of Technology*. La très grande majorité de ces environnements utilisent des instructions en anglais.

Plusieurs de ces langages pour débutants utilisent des interfaces visuelles de programmation dans lesquelles le programmeur crée et manipule des boites logiques qu'il connecte entre elles à l'aide de la souris. Les programmes conçus dans ces environnements sont habituellement des animations ou de petits jeux vidéo simples. Ces langages ciblent habituellement des étudiants plus jeunes que ceux du niveau collégial. Par exemple, « Scratch est spécialement conçu pour les enfants de 8 à 16 ans » (Massachusetts Institute of Technology, s.d.). Dans la francophonie, quelques projets pratiques tels que Clic++ (Québec) et AlgoBox (France) ont été produits dans le passé, mais ils sont très peu utilisés, car leurs qualités éducatives ne sont pas démontrées et ils sont considérés comme désuets.

À notre connaissance, en général, au collégial québécois, les langages de programmation professionnels sont utilisés pour l'apprentissage et l'enseignement de l'algorithmique dans les cours introductifs. Comme cela a été expliqué précédemment, la personne ou l'équipe responsable de choisir le langage de programmation a une lourde responsabilité à porter puisque ce choix a un impact de premier plan sur la réussite des étudiants dans un cours d'introduction à la programmation.

À ce jour, peu de littérature dans le domaine de l'éducation québécois s'intéresse à la question de l'enseignement et de l'apprentissage de l'algorithmique et de la programmation dans le contexte du programme collégial de Techniques de l'informatique et encore moins au choix du langage de programmation utilisé. Au moment de l'écriture de cet essai, le Centre de documentation collégial (CDC) ne répertorie aucun essai PERFORMA et aucune recherche PAREA en lien avec ce sujet.

3. L'OBJECTIF GÉNÉRAL DE RECHERCHE

L'objectif général de cette recherche est de concevoir une grille d'analyse des langages de programmation pour l'apprentissage et l'enseignement de l'algorithmique dans le cadre du programme Techniques de l'informatique au collégial québécois pour permettre à un enseignant, une enseignante ou une équipe de sélectionner le langage de programmation le plus approprié pour leurs étudiants et étudiantes, et pour faciliter leur rapport aux savoirs algorithmiques.

Cette recherche se rattache au pôle « Innovation » de la maitrise en enseignement au collégial offert par le secteur PERFORMA de la Faculté d'éducation de l'Université de Sherbrooke. Le *Guide de présentation du bloc recherche innovation et analyse critique de la maîtrise en enseignement au collégial* (PERFORMA, 2017) le décrit ainsi :

> L'innovation constitue un : « processus délibéré de transformation des pratiques par l'introduction d'une nouveauté curriculaire, pédagogique ou organisationnelle qui fait l'objet d'une dissémination et qui vise l'amélioration durable de la réussite éducative des élèves ou des étudiants » (Conseil supérieur de l'éducation, 2006, p. 26). Conçue dans un contexte donné, l'innovation est documentée et fondée sur un cadre de référence. Elle est validée par les pairs ou par les personnes à qui l'innovation s'adresse ou encore elle fait l'objet d'une expérimentation auprès de ces dernières. (*Ibid.*, p. 24).

DEUXIÈME CHAPITRE
LE CADRE DE RÉFÉRENCE

Ce chapitre présente le cadre de référence utilisé dans cette recherche. Nous estimons que cet essai présente un enjeu de recherche dont la nature est fondamentalement didactique. Selon Vergnaud, la didactique couvre de « l'épistémologie des disciplines aux avancées de la psychologie cognitive [;] c'est l'ensemble du processus construisant le rapport au savoir qui est analysé » (1999, p. 1).

Le cadre de référence de cet essai s'appuie principalement sur le modèle de questionnement didactique au collégial présenté dans Lapierre (2008). Ce modèle structurant est composé de cinq entrées dont trois sont identifiées comme des sources et deux comme des ressources. Les sources sont « les éléments de base qu'utilise l'enseignant pour la planification tant du cours que du programme » (Lapierre, 2008, p. 9) tandis que les ressources « sont des moyens techniques et stratégiques qui permettent de passer de la planification à l'intervention ; autrement dit, ce sont des possibilités d'action » (*Ibid.*, p. 8). L'enseignante ou l'enseignant didacticien se retrouve au centre de ce questionnement puisqu'il ou elle est l'agent principal de l'action didactique.

Premièrement, les entrées de ce modèle de questionnement didactique seront présentées et approfondies une à une et des cadres de référence secondaires y seront intégrés. Nous allons ensuite synthétiser les cadres de référence utilisés pour terminer par l'énoncé des objectifs spécifiques de cette recherche.

Ce chapitre se consacre à dégager, à partir des cadres de références, les considérations qui seront approfondies lors de la conception initiale de la grille d'analyse.

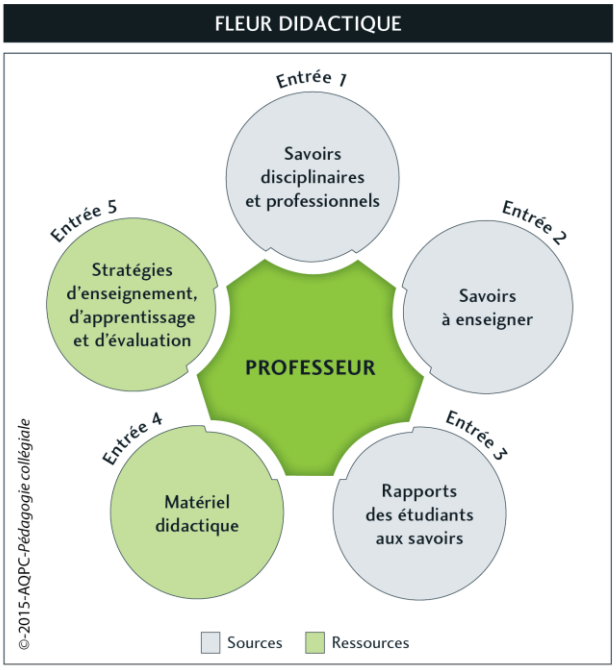


Figure 1 Modèle de questionnement didactique (Prud'homme, 2015, p. 40)

1. LE QUESTIONNEMENT DIDACTIQUE

Le langage de programmation exploité en tant que matériel servant pour l'apprentissage et l'enseignement de l'algorithmique constitue une ressource didactique et fait donc partie de l'entrée 4 du questionnement didactique. Son choix est dépendant non seulement de ses propres propriétés, mais aussi des liens qu'il a avec les autres entrées. C'est pourquoi, dans le cadre de cette recherche, le questionnement didactique, tel que présenté par Lapierre (2008), est utilisé pour dégager différentes considérations (désignées ici comme des *optiques*) fondant des attributs concrets permettant de choisir des langages de programmation en pleine connaissance des causes et des effets sur les actes d'enseignement et d'apprentissage.

1.1 Les savoirs disciplinaires et professionnels

La première entrée du questionnement didactique invite à faire l'inventaire des savoirs disciplinaires et professionnels qui serviront comme point de départ du processus didactique; il s'agit « plus précisément de savoirs savants, de pratiques ou situations professionnelles et de pratiques sociales » (Lapierre, 2008, p. 8).

1.1.1 Les savoirs savants

Un des filons importants à explorer pour cette entrée est constitué des savoirs savants que l'on peut définir comme « les savoirs accrédités par la communauté universitaire et les savoirs provenant de la recherche actuelle » (*Ibid.*, p. 8). Ces savoirs se retrouvent habituellement dans les livres de référence utilisés.

Dans le cadre de cet essai, les savoirs savants utilisés pour la recherche d'attributs sont limités aux savoirs nécessaires à la réussite du premier cours de programmation de la formation Techniques de l'informatique du Cégep de Rimouski. Les optiques suivantes ont été découvertes dans le plan du cours : la structure générale d'un programme, les structures de contrôle, les variables, les types de données, les modules, les paramètres et les variables tableaux (Cégep de Rimouski, 2016, p.3).

1.1.2 Les situations professionnelles

En formation technique, la situation professionnelle « est la référence clé. En effet, les situations professionnelles contiennent en pratique toutes les activités caractéristiques de la profession » (Lapierre, 2008, p. 9).

Le programmeur ou la programmeuse a la responsabilité professionnelle d'« écrire, modifier, intégrer et mettre à l'essai le code informatique pour des applications logicielles » (Gouvernement du Québec, 2013, p. 47). Pour accomplir ses responsabilités, il ou elle doit performer plusieurs activités professionnelles dans des situations différentes à des niveaux d'intensité et de complexité variables, selon son

expérience. Certaines situations professionnelles identifiées ont un impact direct sur le langage de programmation choisi pour des novices.

L'écriture et la modification du code sont communément appelées le codage du programme. Il s'effectue « en fonction du langage de programmation utilisé et de l'architecture sur laquelle va s'exécuter le programme » (*Ibid.*, p. 70). Le codage et les modifications du code s'effectuent dans un éditeur de texte spécialisé qui est utilisé pour entrer les instructions du programme une à une.

Lorsqu'il ou elle s'assure du fonctionnement correct de son programme fraichement créé ou lors d'une maintenance d'applications déjà déployées, le programmeur ou la programmeuse est amené à déboguer son code. Le débogage est ce qui permet la correction des défectuosités d'un programme. L'introduction de bogues lors de l'écriture ou de la modification du code est normale. Par contre, on s'attend à ce que leur nombre soit minimisé le plus possible (Gouvernement du Québec, 2015). Il est impossible d'exiger l'écriture de toutes les bonnes instructions d'un premier coup pour un problème moindrement complexe. Le débogage est donc un aspect fondamental de la programmation.

Dans un contexte professionnel, pour déboguer son programme, on doit pouvoir procéder à des investigations, effectuer la revue de code et exécuter des tests de performance, de sécurité et de charge (*Ibid.*). Le débogueur est l'outil logiciel permettant d'effectuer les investigations nécessaires au débogage.

Les situations professionnelles ont permis d'identifier deux optiques : l'écriture et la modification du code ainsi que le débogage du code.

1.1.3 Les pratiques sociales

Les pratiques sociales renvoient aux « activités ou interventions qui ont cours dans un milieu donné et qui ont besoin d'être comprises et situées » (*Ibid.*, p. 9).

Bien que la programmation soit habituellement considérée comme un travail très individuel, c'est en fait un travail d'équipe dans la grande majorité des cas. En effet, pour un logiciel moindrement complexe, plusieurs personnes travailleront ensemble à sa programmation. Bien que chacune d'elles soit responsable d'une partie du code, elles doivent gérer les intégrations de ces parties dans un tout et cela mène à des interactions sociales entre les personnes impliquées.

De plus, la programmation par binôme (*pair programming*), décrite par Williams *et al.* (2000), consiste à programmer sur un ordinateur sur lequel sont connectés deux écrans, deux claviers et deux souris (tous les systèmes d'exploitation modernes supportent cette fonctionnalité). Une des deux personnes entre des instructions du programme et l'autre les lit. Les interactions sont multiples et la discussion est omniprésente entre les deux personnes. Elles peuvent arrêter temporairement la programmation pour utiliser un tableau blanc et faire des diagrammes, mais seule la personne qui programme peut manipuler l'ordinateur. On s'échange le contrôle à l'improviste ou en suivant des règles, selon la situation et les personnalités impliquées.

La programmation par binôme est bénéfique à l'apprentissage de la programmation selon McDowell *et al*. (2002), Nagappan *et al.* (2003) et Williams et Upchurch (2001). Dans le contexte collégial, la programmation en binôme peut être pratiquée avec d'autres étudiants et étudiantes ou avec l'enseignant ou l'enseignante.

Dérivée de la programmation en binôme, la programmation en aquarium (*fishbowl programming*), décrite par Johansen *et al.* (2002), se pratique en groupe de plus de deux personnes. Une de ces personnes programme sur un ordinateur connecté à un projecteur et les autres regardent le code produit. Ces derniers discutent de l'évolution du programme et lorsqu'ils atteignent un certain consensus, ils en discutent avec la personne qui programme. Ceci permet de filtrer les mauvaises idées avant de

les amener au programmeur ou à la programmeuse qui doit se concentrer pour écrire le programme.

Les pratiques sociales ont permis d'identifier trois optiques : la programmation en équipe, la programmation en binôme et la programmation en aquarium.

1.2 Les savoirs à enseigner

Les savoirs à enseigner constituent la deuxième entrée du questionnement didactique selon Lapierre (2008). Cette étape permet de s'assurer que les savoirs disciplinaires et professionnels identifiés sont en harmonie avec les contraintes établies par des niveaux décisionnels en amont.

Au collégial québécois, « les programmes prescrits par le Ministère (devis ministériel) en formation technique […] sont formulés en termes de finalités, de buts et de compétences (objectifs et standards) » (Ibid., p. 6). Ces finalités sont, pour la très grande majorité, tirées d'une analyse approfondie de la situation professionnelle et de discussions entre employeurs, enseignants et enseignantes ainsi que spécialistes de contenu.

Deux compétences énoncées dans le devis ministériel sont en lien avec l'objectif principal de cette recherche : pouvoir produire des algorithmes et pouvoir exploiter un langage de programmation structurée (Gouvernement du Québec, 2000).

1.2.1 Produire des algorithmes

L'énoncé complet de la compétence « 016W : Produire des algorithmes » provenant du devis ministériel pour le programme de formation Techniques de l'informatique au collégial québécois se trouve à l'annexe A de cet essai. Elle est composée de trois éléments : analyser la situation, mettre au point l'algorithme et valider l'algorithme (*Ibid.*). Chacun de ces éléments de compétence sera une optique

considérée lors de la recherche d'attributs, bien que ces étapes se déroulent habituellement avant l'utilisation d'un langage de programmation.

1.2.2 Exploiter un langage de programmation structurée

La programmation structurée est le type de programmation utilisée pour transposer les algorithmes tels qu'ils sont décrits dans le premier chapitre de cet essai. En effet, « en programmation structurée, pour construire des algorithmes, on se sert des trois structures de base suivantes : la séquence, la répétitive et l'alternative » (Bard, 1991).

Le devis ministériel la mentionne explicitement dans l'énoncé de compétence « 016S : Exploiter un langage de programmation structurée » qui se trouve à l'annexe B de cet essai.

Cet énoncé de compétence est composé de cinq éléments : la préparation de l'environnement de programmation, l'adaptation de l'algorithme aux contraintes du langage de programmation, la traduction de l'algorithme dans le langage de programmation, la compilation du programme et, finalement, sa validation (Gouvernement du Québec, 2000, p. 79). Ici encore, chacun de ces éléments de compétence sera une optique considérée lors de la construction de la grille d'analyse.

1.3 Le rapport des étudiants aux savoirs

Cette entrée du questionnement didactique couvre, entre autres, le patrimoine intellectuel, les valeurs et les préconceptions des étudiants et étudiantes, et, finalement, tout ce qui est en lien avec la relation entre l'apprenant et les savoirs (Lapierre, 2008, p. 9).

Cette recherche abordera quelques aspects du rapport des étudiants aux savoirs, mais cette relation est extrêmement vaste puisque le langage de programmation est l'interface entre le programmeur ou la programmeuse et la machine. Cette entrée du

questionnement didactique aurait avantage à être sujette de recherches plus approfondies.

1.3.1 Les prérequis

Les prérequis de cours sont une façon d'en contrôler l'accès en fonction du patrimoine intellectuel de l'étudiant : son cheminement scolaire fait foi du patrimoine accumulé et construit jusqu'à maintenant.

Bien qu'aucun prérequis en lien avec l'informatique n'existe pour l'inscription au premier cours de programmation du programme de formation, cette recherche considérera les optiques suivantes : la langue maternelle et le bagage en mathématiques du programmeur.

1.3.2 Les erreurs des débutants

Notre recherche s'intéresse aux attributs des langages de programmation lors de leur utilisation dans un contexte éducatif par des débutants. Guibert *et al.* (2005) identifient, à l'aide de recherches antérieures en psychologie cognitive et en didactique, deux grands types d'erreurs affligeant les novices de la programmation.

Le premier type d'erreurs concerne les erreurs dont l'origine dépend de la compréhension qu'ils ou elles ont du modèle d'exécution d'un programme, soit comment l'ordinateur exécute le programme entré. Initialement, la plupart d'entre eux voient l'ordinateur comme une boite noire opaque contenant un esprit magique lui permettant de deviner ce que l'on veut lui faire faire (*Ibid.*, 2005, p. 2). La réalité est bien loin de cela puisque l'ordinateur ne fait qu'exécuter aveuglément ce que le code lui indique de faire. À la base, c'est une machine sans intelligence ni compréhension de ce qui est attendu par l'humain le manipulant.

L'environnement de développement dans lequel se fait la programmation permet, au minimum, d'éditer le code du programme, mais certains environnements permettent aussi d'inspecter son exécution pour en permettre le débogage. Les

fonctionnalités d'inspection du code en exécution disponibles dans l'environnement jouent un rôle primordial à la construction d'un modèle d'exécution correct. En effet, « a good novice programming environment should be able to show students how everything is going on inside, so they could infer a correct execution model. » (*Ibid.*, p. 3)

Le deuxième type d'erreurs des novices regroupe celles dont la source est la distance cognitive entre la représentation naturelle qu'ils ou elles ont d'un concept et la représentation numérique du langage dans lequel ils doivent traduire cette représentation naturelle (*Ibid.*, p. 2). Cette distance cognitive varie en fonction du langage de programmation utilisé puisque chaque langage possède ses propres représentations sémantiques et syntaxiques. Dans un contexte d'apprentissage, on voudra donc réduire à sa valeur minimale cette distance pour permettre l'expression la plus simple possible des opérations que l'étudiant veut transmettre à l'ordinateur.

Chacun des deux types d'erreurs identifiés sera une optique considérée lors de la construction de la grille d'analyse.

1.4 Le matériel didactique

Notre recherche s'intéresse à l'analyse des attributs des langages de programmation et de leurs environnements de développement pour leur utilisation en tant que matériel didactique dans un contexte collégial. Les trois premières entrées du questionnement didactique ont toutes été analysées dans l'optique de leur utilisation dans ce contexte.

Le matériel didactique est un « moyen technique » (Lapierre, 2008), un outil créé ou sélectionné pour les étudiants et étudiantes. Le matériel didactique est fondamentalement une ressource (*Ibid.*, p. 8). Elle peut être plus ou moins abstraite : ce peut être un objet physique, une œuvre, un logiciel, un langage ou juste des données.

1.4.1 Le rapport de l'enseignant au langage de programmation

Pour préparer et donner son cours, l'enseignant ou l'enseignante devra connaitre le langage de programmation sélectionné. S'il produit ses propres notes de cours, il devra pouvoir y intégrer du code pour illustrer ses explications.

En plus des notes de cours, il est possible que l'enseignant ait à produire des programmes exemples, des exercices et des évaluations.

1.4.2 Les attributs des systèmes pour programmeurs débutants

Kelleher et Pausch (2005) ont créé une taxonomie permettant de classer les environnements de programmation facilitant l'apprentissage en fonction des stratégies mises en œuvre. Cette taxonomie est vaste et ne sera pas présentée dans cet essai. Par contre, pour construire leur taxonomie des environnements, ils ont créé une grille d'attributs leur permettant d'analyser ces environnements et leur langage. Nous reprenons cette grille et nous nous permettons de l'intégrer dans notre travail de recherche bien qu'elle soit de haut niveau et ne s'intéresse pas en profondeur aux nuances sémantiques et syntaxiques des langages de programmation.

Les catégories d'attributs identifiées sont : le style de programmation supporté – aussi appelé paradigme de programmation –, les construits supportés, la forme que prend la représentation du code, la façon de construire les programmes, le support pour la compréhension des programmes, la prévention des erreurs de syntaxe, l'accessibilité du langage, les moyens de communication offerts par l'environnement et le choix des tâches à programmer (*Ibid.*, p. 59-61).

Chacune de ces catégories sera une des optiques de questionnement nous permettant d'identifier des attributs lors de la présentation des résultats.

1.4.3 Cadre de référence pour l'évaluation du matériel didactique numérique

Le Ministère de l'Éducation et de l'Enseignement supérieur publie des outils permettant l'évaluation de matériel didactique sur son site web. En grande partie, ces outils sont utilisés par le Ministère lors de l'approbation du matériel didactique pour les niveaux primaire et secondaire tel que le stipule la Loi sur l'instruction publique, et plus précisément l'article 462 : « Le ministre peut établir la liste des manuels scolaires, du matériel didactique ou des catégories de matériel didactique approuvés par lui qui peuvent être choisis pour l'enseignement des programmes d'études qu'il établit » (Gouvernement du Québec, 2010, p.2).

« Le Bureau d'approbation du matériel didactique [(BAMD)] est la structure ministérielle de la Direction des ressources didactiques (DRD) qui, depuis 1980, procède à l'évaluation et à l'approbation du matériel didactique à l'aide de critères approuvés par la ministre » (Gouvernement du Québec, 2010). Un des multiples outils fournis par le BAMD est un cadre de référence pour l'évaluation du matériel didactique numérique, détaillé dans (Gouvernement du Québec, 2016). Pour le moment, aucun outil ministériel d'évaluation du matériel didactique n'existe pour le niveau collégial ; nous allons donc adapter cet outil à notre contexte.

Dans ce cadre de référence, les critères d'évaluation du matériel didactique numérique sont catégorisés en facilitateurs techniques et facilitateurs ergonomiques.

Les facilitateurs techniques permettent de s'assurer que :

- le matériel à consulter directement sur un site Web est accessible avec les principaux navigateurs modernes qui respectent les standards établis ;
- le matériel est fourni dans un standard reconnu […], lisible sur au moins l'un des principaux dispositifs […] et systèmes d'exploitation actuels […] sans limiter les fonctionnalités de navigation et d'accessibilité universelles de ces systèmes ;
- le matériel ne présente aucune erreur structurelle […] ;

- le matériel donne des indications techniques adaptées à la clientèle concernant l'utilisation et le fonctionnement ;
- le matériel donne accès à une fonction « Aide », dans le matériel et en ligne, en cas de problème.

(Gouvernement du Québec, 2016, p. 2)

Tous les critères techniques énumérés par le Ministère sont retenus comme optiques de questionnement pour notre recherche. Bien que certains critères soient en lien direct avec le matériel didactique sur le web, nous les considérons comme admissibles puisque certains environnements de programmation, surtout ceux pour novices, sont disponibles en version web.

Pour leur part, les facilitateurs ergonomiques permettent de s'assurer que :

- des balises de repérage sont présentes (ex. : table des matières, signets, liens internes entre les chapitres […]) ;
- l'interface propose des éléments facilitant la lisibilité du matériel affiché à l'écran […] ;
- la navigation est intuitive, c'est-à-dire que l'interface est constante et logique sur une plateforme donnée et que les utilisateurs ont accès à tous les outils nécessaires pour s'approprier les opérations ;
- les sources des ressources numériques externes présentes dans le matériel sont mentionnées.

(Gouvernement du Québec, 2016, p. 3)

Seulement les deuxième et troisième critères s'appliquent à notre situation puisque les autres sont en lien avec les contenus alors que nous évaluons un outil. Nous retenons donc l'interface facilitant la lisibilité et la navigation intuitive comme des optiques à considérer lors de la création de la grille d'analyse.

1.5 Les stratégies d'enseignement, d'apprentissage et d'évaluation

La dernière entrée du questionnement didactique selon Lapierre (2008) permet au didacticien ou à la didacticienne de se questionner sur les stratégies mises en œuvre lors de la formation. Comme le matériel didactique, c'est une ressource, un moyen d'action (*Ibid.*, p. 8).

Les stratégies déployées pour l'enseignement, l'apprentissage et l'évaluation doivent être cohérentes entre elles et sont habituellement chapeautées par une théorie de l'apprentissage qui s'inscrit elle-même dans un paradigme épistémologique. En effet, le paradigme est un courant qui permet de « guider la démarche de ceux qui y adhèrent » (Ménard et St-Pierre, 2014, p. 20), c'est un « cadre de référence pour guider l'action » (*Ibid.*, p. 21). Formellement, le paradigme « décrit un ensemble de croyances, de valeurs et de pratiques » (*Ibid.*, p. 20). En éducation, deux paradigmes épistémologiques sont habituellement considérés : le paradigme positiviste et le paradigme constructiviste (*Ibid.*, p. 21).

Le paradigme positiviste « considère que la connaissance est un morceau de la réalité qui peut être codée […] indépendamment de celui qui l'observe. Selon les positivistes, la connaissance est construite scientifiquement par des procédés objectifs » (*Ibid.*, p. 21). Une des théories de l'apprentissage qui s'inscrit bien dans ce courant est le béhaviorisme (*Ibid.*, p. 22). Pour ses adhérents, les actes d'enseignement sont vus comme des piliers de l'apprentissage et ces actes sont plus magistraux qu'interactifs. Les connaissances à apprendre sont découpées en petits morceaux présentés aux étudiants et étudiantes. La planification de l'enseignement y joue un grand rôle (*Ibid.*, p. 28) et les évaluations sont multiples, mais petites, pour permettre la rétroaction sur le processus d'acquisition des connaissances.

Le paradigme constructiviste, en rupture avec le paradigme positiviste, « définit la connaissance comme une construction personnelle qui s'appuie sur les

connaissances antérieures de l'individu, ses buts et ses expériences vécues » (*Ibid.*, p. 31). La théorie de l'apprentissage du cognitivisme s'inscrit dans ce paradigme (*Ibid.*, p. 28). Pour ceux qui optent pour cette vision de l'apprentissage, ce sont les activités d'apprentissage qui priment sur l'enseignement (*Ibid.*, p. 31) et les évaluations y sont moins nombreuses, mais plus déterminantes. On favorisera l'approche par problème ou par projet.

Il existe d'autres théories de l'apprentissage telles que la conception humaniste de l'apprentissage et le connectivisme, mais nous nous sommes concentrés sur les deux théories les plus répandues au niveau de la recherche et de l'enseignement au collégial québécois au moment de l'écriture de cet essai.

Ainsi, nous retenons la vision béhavioriste et la vision cognitiviste de l'apprentissage comme des optiques à considérer lors de la création de la grille d'analyse des langages de programmation.

2. SYNTHÈSE DU CADRE DE RÉFÉRENCE : DES OPTIQUES À CONSIDÉRER LORS DE LA CRÉATION DE LA GRILLE D'ANALYSE

Les optiques dégagées dans ce chapitre sont réunies dans la grille ci-dessous. Cette grille servira de point de départ du chapitre 4.

Tableau 3 Les optiques à considérer

Entrée du questionnement didactique selon Lapierre (2008)	**Éléments considérés**	**Optiques à considérer**
1. SAVOIRS DISCIPLINAIRES ET PROFESSIONNELS	1.1 Les savoirs savants selon Lapierre (2008)	La structure générale d'un programme
		Les structures de contrôle
		Les variables
		Les modules
		Les paramètres
		Les variables tableaux
	1.2 Les situations professionnelles selon Lapierre (2008)	Écriture et modification du code selon (Gouvernement du Québec, 2013)
		Débogage du code selon (Gouvernement du Québec, 2015)
	1.3 Les pratiques sociales selon Lapierre (2008)	Programmation en équipe
		Programmation en binôme
		Programmation en aquarium
2. LES SAVOIRS À ENSEIGNER	2.1 Produire des algorithmes selon (Gouvernement du Québec, 2000)	Analyser la situation
		Mettre au point l'algorithme
		Valider l'algorithme
	2.2 Exploiter un langage de programmation structurée selon (Gouvernement du Québec, 2000)	La préparation de l'environnement de programmation
		L'adaptation de l'algorithme aux contraintes du langage de programmation

		La traduction de l’algorithme dans le langage de programmation
		La compilation du programme
		La validation du programme
3. LE RAPPORT DES ÉTUDIANTS AUX SAVOIRS	3.1 Les prérequis	Langue maternelle du programmeur
		Bagage en mathématiques du programmeur
	3.2 Les erreurs des débutants selon Guibert *et al.* (2005)	Erreurs dont l’origine dépend de la compréhension ~~qu’ils ont~~ du modèle d’exécution d’un programme
		Distance cognitive entre la représentation naturelle ~~qu’ils ont~~ d’un concept et la représentation du langage
4. LE MATÉRIEL DIDACTIQUE	4.1 Le rapport de l'enseignant au langage de programmation	La connaissance qu'a l'enseignant ou l’enseignante du langage de programmation
		Production de notes de cours
		Création de programmes exemples
		Création d'exercices
		Création d'évaluations
	4.2 Attributs des systèmes pour programmeurs débutants selon Kelleher et Pausch (2005)	Paradigme de programmation
		Construits supportés
		Représentation du code
		Construction du programme
		Supports améliorant la compréhension de l'exécution du programme

		La prévention des erreurs de syntaxe
		Accessibilité du langage
		Communications
		Choix des tâches
	4.3 Cadre de référence pour l'évaluation du matériel didactique numérique selon (Gouvernement du Québec, 2016)	Site Web accessible avec les principaux navigateurs modernes qui respectent les standards établis
		Fourni dans un standard reconnu lisible sur au moins un des principaux dispositifs et systèmes d'exploitation actuels
		Ne présente aucune erreur structurelle
		Donne des indications techniques adaptées à la clientèle concernant l'utilisation et le fonctionnement
		Donne accès à une fonction « Aide », dans le matériel et en ligne, en cas de problème
		L'interface propose des éléments facilitant la lisibilité du matériel affiché à l'écran
		La navigation est intuitive
5. STRATÉGIES D'ENSEIGNEMENT, D'APPRENTISSAGE ET D'ÉVALUATION	Paradigmes et théories de l'apprentissage selon Ménard et St-Pierre (2014).	Vision béhavioriste de l'apprentissage
		Vision cognitiviste de l'apprentissage

3. LES OBJECTIFS SPÉCIFIQUES

Les personnes responsables de la sélection du langage de programmation utilisé pour l'apprentissage et l'enseignement de l'algorithmique ont une grande responsabilité à assumer et aucun outil n'existe pour l'analyse qui sous-tend nécessairement le choix éclairé. Le cadre de référence développé dans cette recherche permet de discriminer des optiques permettant l'observation d'un langage de programmation sous différents aspects didactiques. Ces optiques seront utiles pour permettre de dégager des attributs concrets des langages de programmation en lien avec chaque optique.

Les objectifs spécifiques de cette recherche sont donc les suivants :

1. Dégager les attributs d'un langage de programmation pour l'apprentissage et l'enseignement de l'algorithmique adapté aux besoins des cours de programmation introductifs en Techniques de l'informatique au niveau collégial québécois.
2. Mettre au point une grille d'analyse des langages de programmation à partir des attributs dégagés.
3. Valider la grille d'analyse auprès de pairs quant à son utilité pour le choix d'un langage de programmation approprié.

TROISIÈME CHAPITRE
LA MÉTHODOLOGIE

Les considérations méthodologiques qui ont guidé cette recherche sont décrites dans ce chapitre. Le devis méthodologique appliqué est directement dérivé du modèle de recherche développement tel que proposé par Loiselle et Harvey (2009). Comme indiqué dans le premier chapitre de cet essai, cette recherche s'inscrit dans le pôle innovation de la Maitrise en enseignement au collégial du secteur PERFORMA de la Faculté d'éducation de l'Université de Sherbrooke et la recherche développement a justement « pour objectif premier de développer des outils matériels ou conceptuels utiles pour agir sur une situation locale donnée » (Loiselle et Harvey, 2007, p. 46).

1. LA RECHERCHE DÉVELOPPEMENT

L'objectif général de cette recherche est de concevoir une grille d'analyse des langages de programmation pour l'apprentissage et l'enseignement de l'algorithmique pour son utilisation dans le programme Techniques de l'informatique au collégial québécois. La recherche développement permet la mise en œuvre d'un processus global de conception et d'amélioration d'un produit ainsi que la description et l'analyse du déroulement de ce processus (Loiselle et Harvey, 2007).

Selon Loiselle et Harvey (2009), la recherche développement se distingue du simple développement de produit grâce à huit caractéristiques qui en font une démarche rigoureuse à forte teneur en scientificité :

1. Le caractère novateur du produit développé ;
2. Une description détaillée du contexte et du déroulement de l'expérience de développement ;
3. Une collecte des données détaillée et à caractère scientifique ainsi qu'une analyse rigoureuse de ces données ;

4. L'établissement de liens entre l'expérience de développement et un ensemble de connaissances ;
5. L'émergence des caractéristiques essentielles du produit à partir de l'expérience ;
6. La justification de toutes les modifications lors de l'élaboration ;
7. L'ouverture vers des pistes de recherche ;
8. L'obligation de diffuser les résultats de la recherche.

À l'aide du modèle de recherche développement sélectionné et en mettant en œuvre rigoureusement chacune des étapes du processus global proposé par ce modèle, notre recherche possède ces huit caractéristiques et se distingue donc du simple développement de produit.

2. L'APPROCHE MÉTHODOLOGIQUE

L'approche méthodologique de cette recherche est dictée par le modèle de la recherche développement sélectionné : « [...] le choix d'une posture interprétative nous apparaît soutenir le mieux le développement du produit et l'induction des principes issus de l'expérience de développement. » (Loiselle et Harvey, 2007, p. 47-48).

En effet, la posture épistémologique interprétative du chercheur lui permet de s'intéresser aux aspects qualitatifs décrivant les perceptions du chercheur lui-même, mais aussi des participants et participantes à la recherche. Cette posture – aussi appelé paradigme naturaliste – admet que la réalité sociale est multiple et construite en fonction des perceptions de chacun (Fortin, 2010). De plus, par l'adoption d'une posture interprétative, notre recherche vise explicitement la transférabilité des résultats et non leur généralisation (PERFORMA, 2017, p. 23).

Loiselle et Harvey (2007) suggèrent explicitement l'approche qualitative pour la recherche développement :

> Si on accepte cette vision privilégiant une posture interprétative pour la recherche développement, on s'orientera vers une démarche principalement qualitative. Les données ne seront pas recueillies principalement pour démontrer l'efficacité du produit, mais pour mettre en évidence les événements sous-tendant les choix faits en cours de développement et réviser au besoin ces choix. (Loiselle et Harvey, 2007, p. 48)

L'approche qualitative « se caractérise par la compréhension des phénomènes et cherche à décrire la nature complexe des êtres humains et la manière dont ils perçoivent leurs propres expériences à l'intérieur d'un contexte social particulier. » (Fortin, 2010, p. 258). Dans le cadre de cet essai, le travail de conception et d'amélioration du produit permettant l'accomplissement de l'objectif de recherche est décrit, expliqué et analysé. Ce travail est exécuté par le chercheur lui-même. L'évolution du projet, étape par étape, avec justifications, permet la compréhension du phénomène de conception de la grille d'analyse des langages de programmation. De plus, l'approche qualitative permettra au chercheur d'entrer en relation avec les participants et participantes à la recherche pour collecter des données qualitatives à propos de l'interaction qu'ils ont eue avec la grille d'analyse.

L'approche qualitative correspond donc au type de méthode de recherche permettant l'atteinte de l'objectif général de cette recherche développement.

3. LE DÉROULEMENT DE LA RECHERCHE

Selon le modèle méthodologique retenu (Loiselle et Harvey, 2009), le déroulement d'une recherche développement est composé de cinq phases macroscopiques et chacune de ces phases contient plusieurs étapes. Ces cinq phases

macroscopiques sont l'origine de la recherche, le référentiel, la méthodologie, l'opérationnalisation et les résultats.

La figure suivante résume bien le déroulement de la recherche développement :

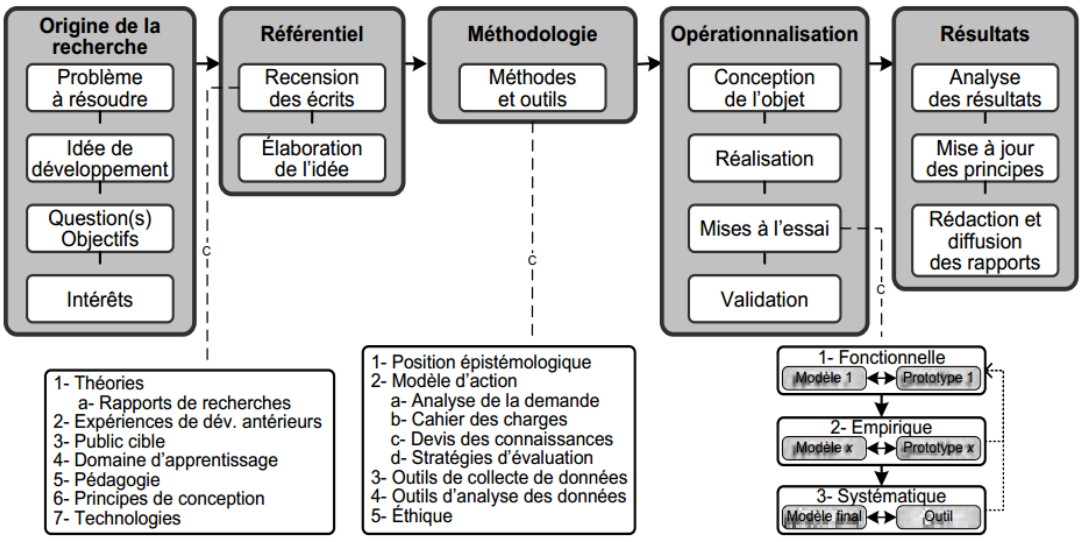


Figure 2 Modèle de recherche développement en éducation (Loiselle et Harvey, 2009, p. 110)

La phase « Origine de la recherche » du modèle est détaillée dans le chapitre 1 de cet essai, la problématique, tandis que la phase « Référentiel » est décrite dans le chapitre 2, le cadre de référence. Le présent chapitre de cet essai, le troisième, décrit la phase « Méthodologie » de la recherche développement. Finalement, les phases « Opérationnalisation » et « Résultats » du modèle de Loiselle et Harvey sont présentés ensemble dans le dernier chapitre de l'essai : la présentation et l'interprétation des résultats.

Bien que l'essai présente la démarche de recherche développement dans un ordre séquentiel et que « la présente modélisation amène à séquencer la démarche, il importe de conserver à l'esprit que l'itération est indissociable d'un tel projet de développement et que des allers et retours constants se font entre les diverses étapes proposées dans le modèle. » (*Ibid.*, p. 113).

3.1 Calendrier de la recherche

Tableau 4 Calendrier de la recherche

Phases	Étapes	Chronologie
Origine de la recherche	Problématique	• Automne 2014 (MEC800) ; • Hiver et automne 2016 (MEC802) ; • Hiver 2017 (MEC803).
Référentiel	Cadre de référence	• Hiver et automne 2016 (MEC802) ; • Hiver 2017 (MEC803).
	Élaboration de l'idée	• 2014 à 2016
Méthodologie	Méthodes et outils	• Hiver 2015 (MEC801) ; • Hiver et automne 2016 (MEC802) ; • Hiver 2017 (MEC803).
Opérationnalisation	Conception initiale	Automne 2017
	Réalisation du premier prototype	Automne 2017
	Sollicitation des experts	Automne 2017
	Mise à l'essai auprès d'experts	Hiver 2018
	Première révision de la conception	Hiver 2018
Résultats	Analyse des résultats	Hiver 2018
	Mises à jour des principes	Hiver 2018
	Rédaction et diffusion des rapports	Automne 2017 et Hiver 2018

4. LA COLLECTE DES DONNÉES

Les outils de collecte des données recueillent, auprès des participants et participantes, les données qui seront analysées pour obtenir des résultats. Débutons par une description de ces personnes pour ensuite s'intéresser spécifiquement aux outils.

4.1 Les participants et participantes à la recherche

Les participants et participantes à la recherche sont le chercheur-développeur et un échantillon de pairs.

Il est attendu dans le cadre de la recherche développement d'avoir recours à un échantillon pour effectuer une mise à l'essai du produit développé. Cet échantillon diffère selon sa représentativité de la population visée par le produit ou encore selon leur expertise du domaine, permettant la détection des lacunes dans la conception et la réalisation (Loiselle et Harvey, 2009).

Pour cette recherche développement, une seule mise à l'essai a été effectuée. Elle s'est faite auprès d'enseignants et enseignantes de cours introductifs de programmation au collégial. Cette mise à l'essai et l'analyse des données recueillies lors de celle-ci ont permis de modifier le prototype pour l'améliorer.

4.1.1 Le chercheur-développeur

Le chercheur-développeur est un seul individu, le maitre d'œuvre de la recherche.

Le chercheur-développeur de cette recherche est un enseignant au collégial québécois en Techniques de l'informatique au Cégep de Rimouski depuis neuf ans. Il y enseigne le premier cours de programmation depuis son arrivée et est passionné par l'apprentissage de la programmation par des novices. Il est aussi un expert du

développement de logiciels avec plus de dix années d'expérience professionnelle avant de son orientation vers l'enseignement.

4.1.2 Les pairs

Peu de critères ont guidé la sélection des pairs pour la première mise à l'essai de cette recherche, mais ils étaient très pointus et restrictifs.

Dans le but de pouvoir commenter les attributs identifiés dans la grille d'analyse des langages de programmation, les pairs recrutés doivent connaitre et comprendre l'algorithmique et la programmation, mais aussi comment elles s'enseignent et s'apprennent. Puisque la grille conçue lors de cette recherche cible une utilisation lors du premier cours de programmation du programme Techniques de l'informatique au collégial québécois, les pairs doivent être des enseignants ou enseignantes donnant présentement ce cours.

L'échantillonnage par choix raisonné est utilisé dans la composition de l'échantillon « pour la sélection de participants possédant les caractéristiques recherchées. » (Fortin, 2010, p. 236). Cette sélection s'effectue par « le chercheur [qui] exerce son jugement sur le choix des personnes aptes à fournir l'information liée au but de l'étude » (*Ibid.*, p. 235).

Fortin (2010) propose quatre stratégies d'échantillonnage pour l'échantillonnage par choix raisonné, dont l'échantillonnage homogène qui permet de sélectionner des individus similaires en fonction des critères établis pour assurer une compréhension en profondeur de l'échantillon et donc du groupe représenté par celui-ci.

Pour cette recherche, il est important de faire émerger une compréhension en profondeur de ce que les participants et participantes ont à dire à propos du prototype avec lequel ils interagissent. Dans le cas des pairs, le fait qu'ils ou elles enseignent la

programmation au collégial augmente la probabilité d'une certaine cohérence dans les commentaires émis par les participants, ce qui permet d'améliorer l'émergence de résultats qualitatifs concrets.

L'échantillonnage par choix raisonné accompagné d'une stratégie d'échantillonnage homogène est donc utilisé dans le cadre de cette recherche.

Puisque les critères de sélection sont pointus, il a été possible d'éliminer une grande partie de la population générale en sélectionnant les sujets que dans un groupe d'individus ayant une plus grande probabilité de répondre aux critères, soit l'ensemble des enseignants et des enseignantes en informatique au collégial.

Les pairs recrutés pour la mise à l'essai sont tous des enseignants ou enseignantes au collégial québécois en Techniques de l'informatique. Plus spécifiquement, ils ou elles doivent avoir une expérience d'au moins trois ans dans l'enseignement des cours d'introduction à la programmation.

Dans le cadre de cette recherche, un cours de programmation se trouvant à l'horaire dans l'un des deux premiers trimestres du programme de formation est considéré comme un cours introductif.

Dans le but d'assurer une certaine triangulation des sources de données, des pairs provenant tous de différents cégeps au Québec ont été sélectionnés. Dans le but d'éviter de possibles biais et les apparences de conflits d'intérêts, tous les pairs proviennent de collèges autres que le Cégep de Rimouski, là où enseigne le chercheur.

La taille de l'échantillon est de quatre individus seulement. Nous avons pourtant déployé des efforts que nous jugeons suffisants lors du recrutement: la publication d'une invitation à participer sur le forum québécois de discussion en ligne pour les enseignants et enseignantes du programme de formation Techniques de l'informatique

et l'envoi d'un courriel de masse à tous les enseignants et enseignantes en informatique du réseau collégial québécois. En raison de sa grande homogénéité, nous espérions que, malgré sa petite taille, l'échantillon puisse nous fournir des données utiles et peut-être même nous permettre d'atteindre la saturation des données. La saturation des données est le point à partir duquel les données recueillies deviennent trop répétitives et qu'elles ne permettent plus l'émergence de nouveaux résultats par l'analyse (Fortin, 2010).

4.2 Le journal de bord

Le premier outil de collecte des données, le journal de bord, recueille des données qualitatives auprès du chercheur-développeur. En effet, le « journal de bord permettra généralement de consigner les décisions prises par le développeur ou l'équipe de développeurs, de même que les raisons motivant les choix effectués. » (Loiselle et Harvey, 2009, p. 112). Cet outil jouera donc un rôle majeur pour assurer la transférabilité de la recherche.

Cette journalisation systématique des décisions de conception et de modification ainsi que leurs justifications se produit tout au long du processus global de développement. La journalisation aura lieu avec une plus grande occurrence lors de la conception initiale et de la mise à l'essai puisque beaucoup de décisions devront être prises lors de ces étapes par le chercheur-développeur (Loiselle et Harvey, 2007).

4.3 La mise à l'essai et l'entrevue semi-dirigée

À l'aide du cadre de référence, le chercheur-développeur crée une version initiale de la grille d'analyse des langages de programmation. C'est cette première version qui est soumise aux pairs lors d'une mise à l'essai. Seule la consigne suivante se trouvait dans le courriel accompagnant la grille :

> La grille en est une d'analyse et non d'évaluation. Il n'y a donc pas de pondération associée aux attributs et vous n'aurez pas de pointage final vous

> indiquant qu'un langage est plus approprié qu'un autre. La diversité des contextes d'application, des considérations, des contraintes et des valeurs guidant la sélection d'un langage de programmation prohibe une approche trop rigide. Il faut donc voir cette grille comme un guide d'analyse vous permettant d'avoir une réflexion didactique sur un ou des langages de programmation.

Suite à leur utilisation de la grille pour l'analyse d'un ou plusieurs langages, les personnes participantes sont soumises à une entrevue pour permettre la collecte des commentaires qu'elles ont à formuler sur leur expérience. Ces données collectées permettront d'améliorer la grille.

L'entrevue est « la principale méthode de collecte des données dans les recherches qualitatives. » (Fortin, 2010, p. 427) et, plus particulièrement, elle est un des outils privilégiés par le modèle de recherche développement de Loiselle et Harvey (2009) pour la cueillette des données lors des mises à l'essai. La très grande majorité des questions d'une entrevue sont ouvertes et amènent le sujet à répondre en profondeur. En effet, l'entrevue consiste « à entrer en contact avec un interlocuteur, à rechercher un accès à l'expérience de l'autre, à cerner ses perspectives au sujet des questions étudiées et à tenter de les comprendre, et ce, d'une façon riche, descriptive, imagée » (Karsenti et Savoie-Zajc, 2011, p. 133). Elle permet donc au chercheur de poser des questions pour préciser ce que voulait exprimer le participant ou la participante s'il y a quelconque ambigüité dans sa compréhension.

Dans le cadre de cette recherche, l'entrevue semi-dirigée est préférable à l'entrevue non dirigée puisque le chercheur a une liste de sujets à aborder en lien avec différents aspects de la grille d'analyse des langages de programmation et de son utilisation. Dans une entrevue semi-dirigée, les questions restent ouvertes, mais il y a un plan d'entrevue que l'intervieweur tente de faire respecter (Fortin, 2010). L'intervieweur peut faire preuve de plus ou moins de flexibilité, dépendant des réponses obtenues. Il se permet toujours de poser des sous-questions pour s'assurer

d'avoir bien compris la réponse à une des questions du plan d'entrevue. Ces sous-questions sont spontanées.

Les entrevues sont menées individuellement, le chercheur en vidéoconférence avec un participant ou une participante. L'entrevue a une durée approximative de 45 minutes. Dans le cadre de cette recherche, le chercheur interviewe tous les pairs recrutés lors de la seule et unique mise à l'essai. Pour plus de détails, le lecteur est invité à consulter le plan de l'entrevue en annexe C.

5. L'ANALYSE DES DONNÉES

Les données recueillies lors de cette recherche sont toutes qualitatives : le contenu du journal de bord et les phrases dites lors des entrevues avec les pairs. Ces données sont le produit des outils sélectionnés pour cette recherche qualitative.

Lors de l'analyse des données qualitatives, « le chercheur s'interroge sur le sens contenu dans les données et fait des allers et retours entre ses prises de conscience, ses vérifications sur le terrain, ce qui lui permet d'amender au besoin sa classification des données. » (Karsenti et Savoie-Zajc, 2011, p. 137).

Le modèle de recherche développement utilisé lors de cette recherche propose une approche inductive pour l'analyse des données recueillies auprès d'un échantillon lors d'une mise à l'essai :

> […] les éléments du cadre théorique sont mis de côté pour le temps de l'analyse, afin de laisser émerger les catégories qui structureront cette analyse. Bien que ce cadre théorique puisse être utile au chercheur dans l'analyse des données qualitatives recueillies et dans les décisions orientant le développement, il ne sera aucunement restrictif. Le corpus théorique existant viendra renforcer ou

> relativiser les données recueillies durant l'expérience et sera pris en compte pour déterminer l'évolution future du produit. (Loiselle et Harvey, 2007, p. 51)

Cette approche est semblable à la stratégie de révision de texte d'analyse des données qualitatives selon laquelle le chercheur analyste fait émerger des catégories riches de sens grâce aux interprétations des données qu'il effectue. (Fortin, 2010).

5.1 Le journal de bord

Puisque le journal de bord est utilisé pour collecter des données auprès du chercheur développeur, son contenu ne sera pas analysé de façon formelle et systématique dans le cadre de notre recherche développement. Il sera plutôt utilisé comme référence par le chercheur pour l'écriture de l'essai et l'essentiel de son contenu se retrouvera dans le chapitre 4 lors de la description de la conception initiale de la grille d'analyse des langages de programmation. De plus, à l'aide du journal de bord, le chercheur détaille toutes les modifications apportées au prototype initial ainsi que toutes les justifications de ces modifications. Ce catalogage systématique des modifications et de leurs justifications peut conduire à des observations et des recommandations éclairantes pour quiconque voudrait reprendre le développement pour l'améliorer ou encore développer un autre produit selon la même approche.

5.2 Les entrevues

Les réponses obtenues par le chercheur lors des entrevues semi-dirigées sont compilées dans une grille de classification permettant de regrouper les réponses communes pour chacune des questions posées. Pour y arriver, les réponses jugées semblables sont amalgamées et résumées itérativement jusqu'à l'obtention du niveau descriptif souhaité. Cette codification est appelée la codification descriptive (Fortin, 2010). Lorsque deux réponses sont regroupées pour ne former qu'une seule phrase les résumant, le nombre de réponses représenté dans cette phrase est noté. Cela permettra de classer en ordre de priorité les éléments qui doivent être modifiés suite à

la mise à l'essai : en effet, plus une donnée qualitative revient souvent lors de l'analyse, plus elle aura de l'importance pour le chercheur développeur puisqu'elle aura été émise plus d'une fois.

Le lecteur est invité à consulter la grille de classification des réponses d'entrevues se retrouvant à l'annexe D de cet essai.

6. MOYENS POUR ASSURER LA RIGUEUR ET LA SCIENTIFICITÉ

Même avec une posture épistémologique interprétative – plutôt subjective –, il est important de porter une attention particulière à la rigueur et à la scientificité. En recherche qualitative, l'utilisation d'une démarche rigoureuse est primordiale si l'on veut garantir la validité des résultats (Fortin, 2010). Cette section discute des moyens pris pour assurer la rigueur et la scientificité de la recherche.

6.1 La rigueur en recherche qualitative

Fortin (2010) propose quatre critères de rigueur pour la recherche qualitative : la crédibilité, la fiabilité, la transférabilité et la confirmabilité. Le chercheur est responsable de la rigueur de sa recherche. Loiselle (2001) apporte quelques pistes concrètes :

> Une présence significative sur le terrain, l'échantillonnage des sujets, l'utilisation de méthodes de triangulation ainsi que l'énoncé et la prise en compte des présupposés et des orientations épistémologiques du chercheur contribuent à l'atteinte de ces [quatre] critères (p. 92).

Les quatre critères de Fortin (2010) sont-ils respectés par cette recherche ? Le critère de crédibilité réfère directement à la quantité et à la qualité des données (Loiselle, 2001). Les données collectées et analysées lors de la recherche doivent permettre de représenter le phénomène étudié avec une exactitude crédible et

raisonnable. La triangulation permet d'augmenter la crédibilité d'une recherche (Fortin, 2010).

Dans cette recherche, l'utilisation d'un échantillon de taille suffisante pour permettre la saturation des données assure une quantité suffisante de données. De plus, la triangulation, assurée par la diversité des lieux de collecte des données, a permis d'assurer la collecte de données de qualité.

Le deuxième critère, la fiabilité, se rapporte à la reproductibilité de la recherche : « on s'assure qu'un autre chercheur, placé dans des circonstances identiques, ferait les mêmes observations » (Fortin, 2010, p. 257).

La fiabilité de cette recherche est assurée par la description complète et détaillée des modifications sur le prototype et de leurs justifications en fonction des données analysées. Ces descriptions permettront de bien expliquer l'évolution du produit développé.

Au premier abord, la fiabilité d'une recherche développement peut paraitre négativement affectée par l'utilisation du journal de bord. Les données qui y sont contenues provenant du chercheur lui-même, les « circonstances identiques » (*Ibid.*) pour assurer la reproductibilité incluent l'utilisation du même chercheur au même moment dans sa vie. Pour pallier ce problème inhérent à la recherche développement menée par un chercheur-développeur, Loiselle (2001) propose que le « chercheur devra analyser et mettre en évidence les éléments (présupposés, valeurs, ou croyances personnelles) influençant l'interprétation des résultats » (Ibid., p. 92).

Cette recherche, à l'aide du journal de bord, détaille toutes les modifications apportées à la version initiale ainsi que toutes les justifications de ces modifications. Ces justifications peuvent provenir des résultats de l'analyse des données ou encore du

chercheur lui-même. Cette explicitation des éléments subjectifs influant la démarche permet de réconcilier la recherche développement et la fiabilité.

Le troisième critère de transférabilité exige que les résultats de la recherche soient transposables, et adaptables si nécessaire, à d'autres contextes ou milieux, par d'autres personnes (Fortin, 2010; Loiselle, 2001). « La pertinence théorique de l'échantillonnage des sujets et une description détaillée du contexte contribuent à la transférabilité d'une recherche » (Loiselle, 2001, p. 91).

La transférabilité de cette recherche est assurée par le contenu de cet essai qui fait la description détaillée de la problématique, du cadre de référence, de la méthodologie, de la présentation et de l'interprétation des résultats. De plus, les références bibliographiques fournies en fin d'essai pourraient permettre à d'autres personnes de s'imprégner de la littérature en amont de cette recherche.

Le dernier critère, la confirmabilité, réfère « à l'objectivité et à la neutralité dans les données » (Fortin, 2010, p. 257). Les catégories et significations qui émergent lors des analyses doivent être vraisemblables, solides et certaines au point où un autre chercheur arriverait à des résultats semblables avec les mêmes données (*Ibid.*).

Dans le cadre de cette recherche, les données recueillies auprès des pairs sont aussi neutres que possible puisqu'ils ou elles n'ont aucun lien personnel ou professionnel avec le chercheur, outre le fait qu'ils ont la même profession que lui.

Finalement, cette démarche de recherche développement respecte les quatre critères de Fortin (2010) et on peut donc la considérer comme rigoureuse.

6.2 La scientificité en recherche développement

Naïvement, on pourrait croire que la recherche développement et le simple développement de produit sont semblables. Par contre, ils sont très différents puisqu'ils ne sont pas de même nature et n'ont pas les mêmes buts. Bien qu'ils produisent tous les deux un objet – le produit – la recherche développement s'intéresse aussi au déroulement du processus de développement de cet objet et à l'analyse de ce déroulement (Loiselle et Harvey, 2009).

Au début de ce chapitre, nous avons introduit huit caractéristiques permettant de distinguer le développement de produit du processus scientifique qu'est la recherche développement : le caractère novateur du produit développé, une description détaillée du contexte et du déroulement de l'expérience de développement, une collecte des données détaillée et à caractère scientifique ainsi qu'une analyse rigoureuse de ces données, l'établissement de liens entre l'expérience de développement et un ensemble de connaissances, l'émergence des caractéristiques essentielles du produit à partir de l'expérience, la justification de toutes les modifications lors de l'élaboration, l'ouverture vers des pistes de recherche et finalement l'obligation de diffuser les résultats de la recherche (Loiselle et Harvey, 2009).

Comme le démontre ce chapitre, l'encadrement méthodologique de cette recherche utilise avec rigueur le modèle de Loiselle et Harvey (2009). De plus, le chercheur est lui-même encadré institutionnellement par le programme de Maitrise en enseignement au collégial dispensé par le secteur PERFORMA de la faculté d'éducation de l'Université de Sherbrooke ainsi que par un directeur d'essai qui est un chercheur accompli en éducation. Ces faits permettent de fournir des garanties quant à la scientificité de la démarche entreprise. De plus, le chercheur, diplômé universitaire en science informatique, s'engage personnellement à garantir une démarche à haute teneur scientifique.

7. ASPECTS ÉTHIQUES DE LA RECHERCHE

L'Énoncé de politique des trois Conseils (EPTC) constitue la norme canadienne en matière d'éthique de la recherche avec des êtres humains (Fortin, 2010).

Tous les participants et participantes à cette recherche ont lu et signé le formulaire de consentement qui leur a été fourni lors de la sollicitation. La lettre d'information relative à la recherche était imbriquée dans ce formulaire, ce qui a permis un consentement libre et éclairé (*Ibid.*). La lettre d'information et le formulaire de consentement sont disponibles à l'annexe E.

Les données collectées lors de la recherche ont été anonymisées lors de l'analyse afin d'assurer le respect de la vie privée et la confidentialité des participants et participantes. Aucune personne participante n'est nommée dans l'essai et aucune information qui permettrait de les identifier n'y est divulguée.

Pour les pairs participant à cette recherche, il n'y a qu'un seul avantage : s'impliquer dans le développement de la grille d'analyse des langages de programmation et y avoir un accès privilégié. Le seul désavantage applicable à tous est l'utilisation d'environ une heure de leur temps pour faire la mise à l'essai et l'entrevue.

Aucun conflit d'intérêt ou apparence de conflit d'intérêts n'a été identifié dans le cadre de cette recherche.

Cette recherche est sous le seuil du risque minimal pour les personnes y participant. L'utilisation des participants et participantes dans cette recherche a été revue et approuvée par le Comité d'éthique de la recherche du secteur PERFORMA de la Faculté d'éducation de l'Université de Sherbrooke ainsi que par les comités d'éthique de recherche de chacun des collèges où enseignent les participants et participantes, s'il en existe un, ou par la direction de l'établissement en question.

QUATRIÈME CHAPITRE
LA PRÉSENTATION ET L'INTERPRÉTATION DES RÉSULTATS

Ce chapitre présente les résultats de la démarche de recherche ainsi que l'interprétation de ces résultats. Rappelons que l'objectif de cette recherche développement est de concevoir une grille d'analyse des langages de programmation pour l'apprentissage et l'enseignement de l'algorithmique dans le programme de formation Techniques de l'informatique au collégial québécois. Cette grille permettra à toute personne responsable de la sélection du langage de programmation utilisé dans les cours introductifs du programme de formation de se fonder sur des critères rationnels lors du processus de décision. La première étape de conception a utilisé le questionnement didactique selon Lapierre (2008) pour extraire différentes considérations de nature didactique, appelées *optiques* dans cette recherche. Les différentes optiques ainsi produites sont ensuite utilisées pour dériver les attributs concrets des langages de programmation qui constitueront la grille d'analyse.

Dans ce chapitre, les stratégies utilisées pour dériver les attributs à partir des optiques sont explicitées. La première version de la grille contenant ces attributs est ensuite présentée. C'est cette première version complète qui fut présentée aux pairs sélectionnés pour une étape de validation. Les données ainsi collectées et leur analyse sont décrites dans ce chapitre suite à la présentation de la première version de la grille. Cette analyse permet la création d'une version améliorée de la grille – le produit final de cette recherche développement – qui sera présentée.

1. LA DÉRIVATION DES ATTRIBUTS

Chaque optique identifiée à l'aide du cadre de référence de questionnement didactique est utilisée pour la dérivation d'attributs concrets et observables des langages de programmation.

Les différentes stratégies de raisonnement utilisées lors de la dérivation des attributs sont identifiées dans le tableau suivant. Il est important de noter que lorsque la stratégie réfère à l'acte d'inventorisation, il s'agit d'un inventaire fait par le chercheur et est donc subjectif et non exhaustif.

Tableau 5 Stratégies de dérivation des attributs

Optiques considérées	**Stratégies de dérivation des attributs**	**Attributs**
La structure générale d'un programme	Inventorisation des attributs relatifs à la structure générale d'un programme	Fonction principale (point d'entrée unique)
		Point(s) de sortie unique ou multiples
		La non-utilisation des variables globales
Les structures de contrôle	Inventorisation des structures de contrôle utilisées en programmation	Support pour les séquences
		Support pour les alternatives: *if, else, else if, switch-case*
		Support des répétitives: *while, for, do-while, for each*
Les variables	Inventorisation des options à considérer lors de la déclaration des variables	Syntaxe de déclaration des variables
		Contraintes relatives au nommage des variables
		Typage statique (types acquis à la compilation) versus Typage dynamique (types acquis à l'exécution)

	Inventorisation des options à considérer lors de l'utilisation des variables	Fortement typé (peu de conversions ou transtypages implicites) versus Faiblement typé (beaucoup de conversions et transtypages implicites)
Les modules	Inventorisation des types de modules	Support pour les procédures ou fonctions sans valeur de retour
		Support pour les fonctions
Les paramètres	Inventorisation des types de paramètres	Support pour paramètres par valeur
		Support pour paramètres par référence
Les variables tableaux	Inventorisation des types de tableaux	Support pour tableaux unidimensionnels
		Support pour tableaux multidimensionnels
	Inventorisation des contraintes d'utilisation	Indices de tableau débutant à zéro, un ou au choix.
		Contraintes associées au passage des tableaux en paramètre
Écriture et modification du code selon (Gouvernement du Québec, 2013)	Inventorisation des fonctionnalités importantes d'un éditeur	Éditeur intégré à l'environnement de développement
		Éditeur : support pour copier-coller
		Éditeur : support pour la coloration syntaxique
		Éditeur : support pour le complètement automatique

Débogage du code selon (Gouvernement du Québec, 2015)	Inventorisation des fonctionnalités importantes d'un débogueur	Débogueur intégré à l'environnement de développement
		Débogueur : point d'arrêt sur instruction
		Débogueur : l'inspection des variables
		Débogueur : exécution contrôlée supportant le pas-à-pas et le pas-à-pas approfondi
Programmation en équipe	Inventorisation des fonctionnalités importantes d'un environnement de développement approprié pour la programmation en équipe	Environnement de développement intégré supportant la mise en commun de code source
		Environnement de développement intégré supportant la communication publique et privée entre les étudiants et étudiantes, et avec l'enseignant
Programmation en binôme ou en aquarium	Inventorisation des fonctionnalités importantes d'un éditeur approprié pour la programmation en binôme ou en aquarium	Éditeur : support pour l'affichage des numéros de ligne
Analyser la situation	Inventorisation des fonctionnalités du langage de programmation en lien avec la production d'algorithmes en pseudocode.	Aucun
Mettre au point l'algorithme		Aucun
Valider l'algorithme		Aucun
La préparation de l'environnement de programmation	Inventorisation des fonctionnalités importantes d'un environnement de programmation	Existence de raccourcis clavier
		Support pour l'utilisation des mnémoniques clavier
		Support pour la personnalisation des raccourcis clavier

		Support pour la personnalisation de la police de caractère utilisée dans l'éditeur
		Support pour la personnalisation des couleurs du texte et du fond d'écran dans l'éditeur
		Surbrillance de la syntaxe adaptée au langage de programmation
L'adaptation de l'algorithme aux contraintes du langage de programmation	Inventorisation des contraintes imposées par le langage de programmation pouvant influencer l'adaptation d'algorithmes écrits en pseudocode.	Système de types de données le plus naturel possible
		Opérations d'entrées et sorties simplifiées
		Support pour toutes les structures de contrôle utilisées en algorithmique
		Syntaxe confortable se rapprochant du langage naturel
		Règles sémantiques et paradigmes appropriés pour un cours introductif.
La traduction de l'algorithme dans le langage de programmation	Inventorisation des contraintes imposées par le langage de programmation pouvant influencer l'adaptation d'algorithmes écrits en pseudocode.	Aucun
La compilation du programme	Inventorisation des fonctionnalités importantes d'un compilateur	Compilation automatique lors de l'exécution
		Repérage facile des erreurs de compilation

		Avertissements pertinents ou configurables
La validation du programme	Inventorisation des fonctionnalités importantes d'un environnement de développement pour la validation du programme	Support intégré pour les tests unitaires
Langue maternelle de la programmeuse ou du programmeur	Inventorisation des attributs d'un langage de programmation relatif à la langue maternelle	Langue utilisée pour les mots-clés du langage de programmation
Patrimoine intellectuel en mathématiques de la programmeuse ou du programmeur	Inventorisation des attributs d'un langage de programmation relatif au bagage mathématique	Opérateurs mathématiques familiers
Erreurs dont l'origine dépend de la compréhension qu'ils ont du modèle d'exécution d'un programme	Inventorisation des fonctionnalités permettant d'améliorer la compréhension qu'ils ont du modèle d'exécution d'un programme	Historique des valeurs de variables
		Surbrillance des variables lors du changement de valeur
		Pile des appels facile d'approche
		Valeurs de retour des fonctions présentées dans l'outil d'inspection des variables
Distance cognitive entre la représentation naturelle qu'ils ont d'un concept et la représentation du langage	La distance cognitive est fonction du patrimoine intellectuel	Voir 3.1

La connaissance qu'a l'enseignante ou enseignant du langage de programmation	La connaissance est mesurée qualitativement par le niveau de compétence.	Niveau de compétence de l'enseignante ou enseignant dans ce langage
Production de notes de cours	Inventorisation des fonctionnalités nécessaires à la production de notes de cours	Langage facilement incorporable dans un document Word, OpenOffice ou LaTeX
Création de programmes exemples	Inventorisation des fonctionnalités nécessaires pour la création de programmes exemples	Code des exemples est autosuffisant (isolé, sans dépendances)
		Support pour les commentaires
		Support pour les commentaires multilignes
Création d'exercices	Inventorisation des fonctionnalités nécessaires pour la création d'exercices	L'environnement de développement permet le contrôle des entrées à l'aide de fichiers de donnée pour permettre l'autocorrection.
Création d'évaluations	Inventorisation des fonctionnalités nécessaires pour la création d'évaluations	L'environnement de développement permet le contrôle des entrées et la validation des sorties à l'aide de fichiers de donnée pour permettre la correction automatisée.
Paradigme de programmation	Sélection tirée directement de (Kelleher et Pausch, 2005)	Procédurale (structurée)
		Fonctionnelle
		Orientée-objet
		Événementielle
		Automate à états

Construits supportés	Sélection tirée directement de (Kelleher et Pausch, 2005)	Alternative *(if, switch)*
		Répétitive *(for, while, for each, do…while)*
		Variables
		Paramètres
		Procédures, fonctions et/ou méthodes
		Types personnalisés
		Pré et post conditions
Représentation du code	Sélection tirée directement de (Kelleher et Pausch, 2005)	Texte
		Images
		Organigramme
		Animations
		Formulaires
		Automate fini
		Objets physiques
Construction du programme	Sélection tirée directement de (Kelleher et Pausch, 2005)	Entrée de code
		Assemblage d'objets graphiques
		Démonstration d'actions
		Sélection et remplissage de formulaires
		Assemblage d'objets physiques
Supports améliorant la compréhension de l'exécution du programme	Sélection tirée directement de (Kelleher et Pausch, 2005)	Débogage
		Interprétation physique
		Modifications en direct lors de l'exécution

		Génération d'exemples
La prévention des erreurs de syntaxe	Sélection tirée directement de (Kelleher et Pausch, 2005)	Potentialité des formes physiques
		Sélection à partir des options valides seulement
		Édition dirigée par la syntaxe
		Déposer seulement aux endroits valides
		Meilleurs messages d'erreurs de syntaxe
Accessibilité du langage	Sélection tirée directement de (Kelleher et Pausch, 2005)	Limitation du domaine d'application
		Sélection de mots-clés centrés sur l'utilisateur
		Retirer la ponctuation superflue
		Utilisation du langage naturel
		Élimination des redondances
Communications	Sélection tirée directement de (Kelleher et Pausch, 2005)	Côte à côte
		Manipulation ensemble par réseau
		Partage des résultats par réseau
Choix des tâches	Sélection tirée directement de (Kelleher et Pausch, 2005)	Divertissantes
		Utiles
		Éducatives
Site Web accessible avec les principaux navigateurs modernes qui respectent les standards établis	Inventorisation des fonctionnalités standardisées pour les environnements web	HTML et CSS
		N'utilisant pas les technologies propriétaires

Fourni dans un standard reconnu lisible sur au moins l'un des principaux dispositifs et systèmes d'exploitation actuels	Inventorisation des plateformes logicielles et matérielles	Disponible Windows, Linux ou MacOS
		Disponible sur PC ou Mac
		Disponible en version web
Ne présente aucune erreur structurelle	Inventorisation des erreurs structurelles pouvant possiblement affecter la programmation	Aucun bogue flagrant
		Les menus provoquent tous une opération
		Aucune erreur structurelle dans la documentation
Donne des indications techniques adaptées à la clientèle concernant l'utilisation et le fonctionnement	Inventorisation des indications techniques relatives à la programmation	Messages d'erreur courts
		Messages d'erreur compréhensibles
		Système de complétion automatique pour le code source
Donne accès à une fonction "Aide", dans le matériel et en ligne, en cas de problème	Inventorisation des fonctionnalités nécessaires pour la fonction d'aide	Aide contextuelle disponible
		Index des rubriques d'aide
		Outil de recherche dans les rubriques d'aide
		Aide disponible en ligne
		Aide disponible hors ligne (sans connexion internet)
L’interface propose des éléments facilitant la lisibilité du matériel affiché à l’écran	Inventorisation des fonctionnalités permettant d'améliorer la lisibilité du matériel affiché à l'écran	Choix de la police de caractère et de sa taille
		Choix des couleurs
		Permet le zoom
La navigation est intuitive		Interface avec prise en main facile

	Inventorisation des caractéristiques rendant la navigation intuitive	Interface réutilisant des paradigmes établis
		Appropriation facile par l'utilisateur des opérations du logiciel
Vision béhavioriste de l'apprentissage	Adéquation des types d'activités pédagogiques favorisées et de la vision de l'apprentissage	Facilitant les exercices pratiques et les examens théoriques
Vision cognitiviste de l'apprentissage	Adéquation des types d'activités pédagogiques favorisées et de la vision de l'apprentissage	Facilitant les projets

2. LA VERSION INITIALE DE LA GRILLE D'ANALYSE

Cette section présente la version initiale de la grille d'analyse des langages de programmation : celle soumise aux pairs pour la validation.

Tableau 6 Version initiale de la grille d'analyse des langages de programmation

Entrée du questionnement didactique selon (Lapierre, 2008)	**Éléments considérés**	**Optiques considérées**	**Attributs**	**Langage de programmation nº1**	**Langage de programmation nº2**
1. SAVOIRS DISCIPLINAIRES ET PROFESSIONNELS	1.1 Les savoirs savants selon (Lapierre, 2008)	La structure générale d'un programme	Fonction principale (point d'entrée unique)		
			Point(s) de sortie unique ou multiple		
			La non-utilisation des variables globales		
		Les structures de contrôle	Support pour les séquences		
			Support pour les alternatives: if, else, else if, switch-case		
			Support des répétitives: while, for, do-while, for each		
		Les variables	Syntaxe de déclaration des variables		
			Contraintes relatives au nommage des variables		

			Typage statique (types acquis à la compilation) versus Typage dynamique (types acquis à l'exécution)		
			Fortement typé (peu de conversions ou transtypages implicites) versus Faiblement typé (beaucoup de conversions et transtypages implicites)		
		Les modules	Support pour les procédures ou fonctions sans valeur de retour		
			Support pour les fonctions		
		Les paramètres	Support pour paramètres par valeur		
			Support pour paramètres par référence		
		Les variables tableaux	Support pour tableaux unidimensionnels		
			Support pour tableaux multidimensionnels		
			Indices de tableau débutant à zéro, un ou au choix.		

			Contraintes associées au passage des tableaux en paramètre		
	1.2 Les situations professionnelles selon (Lapierre, 2008)	Écriture et modification du code selon (Gouvernement du Québec, 2013)	Éditeur intégré à l'environnement de développement		
			Éditeur : support pour copier-coller		
			Éditeur : support pour la coloration syntaxique		
			Éditeur : support pour le complètement automatique		
		Débogage du code selon (Gouvernement du Québec, 2015)	Débogueur intégré à l'environnement de développement		
			Débogueur : point d'arrêt sur instruction		
			Débogueur : l'inspection des variables		
			Débogueur : exécution contrôlée supportant le pas-à-pas et le pas-à-pas approfondi		
	1.3 Les pratiques sociales	Programmation en équipe	Environnement de développement intégré		

	selon (Lapierre, 2008)		supportant la mise en commun de code source		
			Environnement de développement intégré supportant la communication publique et privée entre les étudiants et étudiantes, et avec l'enseignant		
		Programmation en binôme ou en aquarium	Éditeur : support pour l'affichage des numéros de ligne		
2. LES SAVOIRS À ENSEIGNER	2.1 Produire des algorithmes selon (Gouvernement du Québec, 2000)	Analyser la situation	Aucun		
		Mettre au point l'algorithme	Aucun		
		Valider l'algorithme	Aucun		
	2.2 Exploiter un langage de programmation structurée selon (Gouvernement du Québec, 2000)	La préparation de l'environnement de programmation	Existence de raccourcis clavier		
			Support pour l'utilisation des mnémoniques clavier		
			Support pour la personnalisation des raccourcis clavier		
			Support pour la personnalisation de la		

			police de caractère utilisée dans l'éditeur		
			Support pour la personnalisation des couleurs du texte et du fond d'écran dans l'éditeur		
			Surbrillance de la syntaxe adaptée au langage de programmation		
		L’adaptation de l’algorithme aux contraintes du langage de programmation	Système de types de données le plus naturel possible		
			Opérations d'entrées et sorties simplifiées		
			Support pour toutes les structures de contrôle utilisées en algorithmique		
			Syntaxe confortable se rapprochant du langage naturel		
			Règles sémantiques et paradigmes appropriés pour un cours introductif.		
		La traduction de l’algorithme dans le langage de programmation	Aucun		

		La compilation du programme	Compilation automatique lors de l'exécution		
			Repérage facile des erreurs de compilation		
			Avertissements pertinents ou configurables		
		La validation du programme	Support intégré pour les tests unitaires		
3. LE RAPPORT DES ÉTUDIANTS AUX SAVOIRS	3.1 Les prérequis	Langue maternelle de la programmeuse ou du programmeur	Langue utilisée pour les mots-clés du langage de programmation		
		Patrimoine intellectuel en mathématiques de la programmeuse ou du programmeur	Opérateurs mathématiques familiers		
	3.2 Les erreurs des débutants selon (Guibert et al., 2005)	Erreurs dont l'origine dépend de la compréhension qu'ils ont du modèle d'exécution d'un programme	Historique des valeurs de variables		
			Surbrillance des variables lors du changement de valeur		
			Pile des appels facile d'approche		
			Valeurs de retour des fonctions présentées dans l'outil d'inspection des variables		

		Distance cognitive entre la représentation naturelle qu'ils ont d'un concept et la représentation du langage	Voir 3.1		
4. LE MATÉRIEL DIDACTIQUE	4.1 Le rapport de l'enseignant au langage de programmation	La connaissance qu'a l'enseignante ou enseignant du langage de programmation	Niveau de compétence de l'enseignante ou enseignant dans ce langage		
		Production de notes de cours	Langage facilement incorporable dans un document Word, OpenOffice ou LaTeX		
		Création de programmes exemples	Code des exemples est autosuffisant (isolé, sans dépendances)		
			Support pour les commentaires		
			Support pour les commentaires multilignes		
		Création d'exercices	Environnement de développement permet le contrôle des entrées à l'aide de fichiers de donnée		

			pour permettre l'autocorrection.		
		Création d'évaluations	Environnement de développement permet le contrôle des entrées et la validation des sorties à l'aide de fichiers de donnée pour permettre la correction automatisée.		
	4.2 Attributs des systèmes pour programmeurs débutants selon (Kelleher et Pausch, 2005)	Paradigme de programmation	Procédurale (structurée)		
			Fonctionnelle		
			Orientée-objet		
			Événementielle		
			Automate à états		
		Construits supportés	Alternative (if, switch)		
			Répétitive (for, while, for each, do…while)		
			Variables		
			Paramètres		
			Procédures, fonctions et/ou méthodes		
			Types personnalisés		
			Pré et post conditions		
		Représentation du code	Texte		
			Images		

			Organigramme		
			Animations		
			Formulaires		
			Automate fini		
			Objets physiques		
		Construction du programme	Entrée de code		
			Assemblage d'objets graphiques		
			Démonstration d'actions		
			Sélection et remplissage de formulaires		
			Assemblage d'objets physiques		
		Supports améliorant la compréhension de l'exécution du programme	Débogage		
			Interprétation physique		
			Modifications en direct lors de l'exécution		
			Génération d'exemples		
		La prévention des erreurs de syntaxe	Potentialité des formes physiques		
			Sélection à partir des options valides seulement		
			Édition dirigée par la syntaxe		
			Déposer seulement aux endroits valides		

			Meilleurs messages d'erreurs de syntaxe		
		Accessibilité du langage	Limitation du domaine d'application		
			Sélection de mots-clés centrés sur l'utilisateur		
			Retirer la ponctuation superflue		
			Utilisation du langage naturel		
			Élimination des redondances		
		Communications	Côte à côte		
			Manipulation ensemble par réseau		
			Partage des résultats par réseau		
		Choix des tâches	Divertissantes		
			Utiles		
			Éducatives		
	4.3 Cadre de référence pour l'évaluation du matériel didactique numérique	Site Web accessible avec les principaux navigateurs modernes qui respectent les standards établis	HTML et CSS		
			N'utilisant pas les technologies propriétaires		

	selon (Gouvernement du Québec, 2016)	Fourni dans un standard reconnu lisible sur au moins l'un des principaux dispositifs et systèmes d'exploitation actuels	Disponible Windows, Linux ou MacOS		
			Disponible sur PC ou Mac		
			Disponible en version web		
		Ne présente aucune erreur structurelle	Aucun bogue flagrant		
			Les menus provoquent tous une opération		
			Aucune erreur structurelle dans la documentation		
		Donne des indications techniques adaptées à la clientèle concernant l'utilisation et le fonctionnement	Messages d'erreur courts		
			Messages d'erreur compréhensibles		
			Système de complétion automatique pour le code source		
		Donne accès à une fonction "Aide", dans le matériel et en ligne, en cas de problème	Aide contextuelle disponible		
			Index des rubriques d'aide		
			Outil de recherche dans les rubriques d'aide		
			Aide disponible en ligne		

			Aide disponible hors ligne (sans connexion internet)		
		L'interface propose des éléments facilitant la lisibilité du matériel affiché à l'écran	Choix de la police de caractère et de sa taille		
			Choix des couleurs		
			Permet le zoom		
		La navigation est intuitive	Interface avec prise en main facile		
			Interface réutilisant des paradigmes établis		
			Appropriation facile par l'utilisateur des opérations du logiciel		
5. STRATÉGIES D'ENSEIGNEMENT, D'APPRENTISSAGE ET D'ÉVALUATION	Paradigmes et théories de l'apprentissage selon (Ménard et St-Pierre, 2014).	Vision béhavioriste de l'apprentissage	Facilitant les exercices pratiques et les examens théoriques		
		Vision cognitiviste de l'apprentissage	Facilitant les projets		

3. LES DONNÉES COLLECTÉES

Cette section résume les propos collectés auprès des quatre personnes participantes composant l'échantillon de la recherche.

Personne participante nº1

La personne participante nº1 s'est retirée de la recherche lors de la phase d'utilisation de la grille par l'envoi d'un courriel contenant le message suivant :

> *Honnêtement je n'ai pas compris comment utiliser la grille. Je m'attendais à ce qu'elle contienne des questions ou des critères, mais je n'y ai rien trouvé de tel. Ça me laisse plutôt dépourvue pour te répondre.*

Nous avons insisté poliment pour discuter avec la personne participante par vidéoconférence pour obtenir plus de détails, mais elle refusa et nous indiqua d'utiliser sa rétroaction par courriel, cité ci-dessus, pour la recherche.

Personne participante nº2

La personne participante nº2 a appliqué la grille d'analyse aux langages de programmation Java et C++. La discussion par vidéoconférence a duré environ 30 minutes. Elle mentionna que la grille était très complète, mais qu'elle couvrait beaucoup plus large que ses besoins puisqu'elle ne se pose pas des questions aussi pointues lors de la sélection d'un langage de programmation.

Elle mentionna les attributs en lien avec les modules et les types de paramètres comme étant très pertinents pour un cours introductif.

Par contre, les attributs tirés de la taxonomie des systèmes pour programmeurs débutants (Kelleher et Pausch, 2005), l'élément 4.1 de la grille, lui ont semblé inutiles

et obscurs. Suite à une explication de ce qu'ils sont, elle mentionna qu'ils pourraient être utiles à quelqu'un, mais pas à elle.

Le format et l'aspect esthétique de la grille lui sont acceptables, mais elle recommande de mettre un texte explicatif dans le haut de la grille pour expliquer comment l'utiliser puisqu'elle est intimidante au premier coup d'œil.

Finalement, la personne mentionne qu'il pourrait être utile de séparer la grille en deux : une adressant les attributs du langage de programmation et l'autre l'environnement de développement : le but étant de réduire la taille de la grille.

Personne participante nº3

La personne participante nº3 s'est retirée de la recherche lors de la phase d'utilisation de la grille par l'envoi d'un courriel contenant le message suivant :

> *Je suis désolé, je vais me retirer de l'étude. Cependant, j'ai utilisé ta grille et voici mes impressions :*
>
> *Comme mise en contexte, pour le premier cours de programmation de 90 heures, nous programmons actuellement en C# avec Visual Studio et une sortie console. Nous utilisons aussi MonoGames pour réaliser un jeu 2D et la librairie MonoBrick pour programmer les robots EV3.*
>
> *Nous désirons lâcher C#, nous hésitons actuellement entre 3 langages pour le premier cours de programmation : JavaScript Python et Java. Avec comme IDE les produits de JetBrains (PyCharm ou PHPStorm ou IntelliJ).*
>
> *Je trouve que ta grille est très précise et a aidé à mes réflexions à un niveau plus théorique et poussé du langage de programmation et de IDE*

> *[environnement intégré de développement]. Cependant elle cible énormément des éléments pointus de ces langages, dont plusieurs ne s'appliquent pas au premier cours de programmation. Par exemple, nous ne passons rien en paramètre, encore moins avec des notions de références et valeurs.*

Nous avons insisté poliment pour discuter avec la personne participante par vidéoconférence pour obtenir plus de détails, mais elle refusa et nous indiqua d'utiliser sa rétroaction par courriel, cité ci-dessus, pour la recherche.

Personne participante nº4

La personne participante nº4 a appliqué la grille d'analyse au langage de programmation Python. La discussion par vidéoconférence a duré environ 40 minutes.

Le premier commentaire mentionné par la personne participante a été que la grille contient trop d'attributs et qu'elle contient tellement d'information qu'elle est décourageante.

Les attributs de la grille les plus pertinents mentionnés par cette personne sont tous ceux en lien avec le débogueur, la « langue utilisée pour les mots-clés du langage de programmation » et le « niveau de compétence de l'enseignante ou enseignant dans ce langage ».

Les attributs de la grille les moins pertinents mentionnés par cette personne sont les types de paramètres puisque, pour elle, il ne s'agit pas d'un contenu de niveau introductif et « tout ce qui est trop générique et répandu » puisque si la réponse est la même pour presque tous les langages, cela ne permet pas de discriminer et que l'attribut est donc inutile. Elle mentionne aussi comme non pertinents : les indices de tableaux débutants à 0 ou à 1 et tout ce qui est en lien avec le matériel didactique puisque

l'enseignant doit se concentrer sur l'acte de programmation et gérer les contraintes en lien avec le matériel didactique comme des conséquences mineures de son choix.

La personne participante nº4 mentionne que la mise en forme de la grille gagnerait en clarté par l'introduction d'instructions d'utilisation. De plus, les deux premières colonnes de la grille lui semblent inutiles : elles pourraient être éliminées et on pourrait ajouter une colonne exemple pour un langage de programmation répandu, ce qui permettrait de comprendre plus facilement comment utiliser la grille. La grille pourrait aussi inclure un ou quelques exemples de programmes simples classiques tels que le calcul itératif de la factorielle d'un nombre pour permettre de comparer du code d'un point de vue plus holistique.

Finalement, la personne participante nº4 mentionne que si la grille ne peut être réduite en taille, il faudrait considérer de la couper en deux pour permettre une utilisation facile : une version abrégée et une version plus complète pour « ceux qui en ont vraiment besoin ».

4. L'ANALYSE DES DONNÉES COLLECTÉES

L'analyse des données se fera à l'aide de la grille de classification des réponses présentée à l'annexe D.

La première question, « Sur quel(s) langage(s) de programmation avez-vous appliqué la grille d'analyse ? », est utile pour la mise en contexte de certaines réponses données par les personnes participantes. Les réponses à cette question ne sont donc pas directement utiles pour la création d'une version améliorée de la grille.

La deuxième question, « Quel(s) manque(s) avez-vous identifié(s) dans la grille ? », est riche en réponses convergentes. Lors de la classification des réponses pour cette question, nous avons pris la liberté d'interpréter subjectivement les commentaires

suivants comme une indication que la grille est trop longue ou trop complexe : « je n'ai pas compris comment utiliser la grille » (personne participante nº1) et « elle cible énormément des éléments pointus de ces langages » (personne participante nº3). Nous utiliserons les deux réponses convergentes à cette question pour améliorer la grille : elle devrait avoir des instructions d'utilisation et elle est trop longue ou trop complexe.

La troisième question, « Quels attributs de la grille vous ont semblé les plus pertinents ? », n'apporte que des réponses hétéroclites. En effet, aucune des réponses à cette question n'est partagée par plus d'un participant. Nous laissons les réponses à cette question de côté, sauf pour balancer les réponses de la question suivante. En effet, la perception positive de la pertinence des attributs ne servira qu'à s'assurer de ne pas retirer un attribut pertinent pour une partie de l'échantillon.

La quatrième question, « Quels attributs de la grille vous ont semblé les moins pertinents ? », apporte aussi un lot de réponses hétéroclites sauf pour une, mentionnée par deux participants : les attributs en lien avec les types de paramètres. Puisque ces attributs en lien avec les types de paramètres ont aussi été mentionnés par un participant comme étant des plus pertinents, nous devons conclure que ces attributs sont pertinents pour certains enseignants ou enseignantes et non pertinents pour d'autres. Nous usons donc de prudence et ne prendrons donc pas en compte cette réponse. Toutes les autres réponses à cette question n'ont été mentionnées qu'une seule fois, mais puisque le commentaire faisant le plus l'unanimité est que la grille est trop longue ou trop complexe, nous allons tous les retirer de la version améliorée de la grille, sauf pour l'entrée 4 du questionnement didactique, le matériel didactique, que nous décidons de ne pas retirer en bloc pour préserver l'intégrité du cadre de référence, bien que cela fût mentionné par la personne participante nº4. Par contre, dans un effort de réduction, l'élément 4.2 du cadre de référence – les attributs des systèmes pour programmeurs débutants selon Kelleher et Pausch (2005) – pourrait être retiré de la grille dans le but d'alléger les attributs associés à l'entrée sur le matériel didactique et permettre aux attributs en lien avec la programmation de prendre plus de place, comme l'a exprimé

cette personne participante qui suggérait d'enlever l'entrée 4 complètement. Bien que l'élément 4.2 soit une partie entière du cadre de référence utilisé dans cette recherche, c'est un élément provenant d'une autre recherche sur les langages de programmation pour débutants et dont les attributs ont directement été greffés dans notre grille. De plus, aucun des attributs de cet élément n'a été mentionné comme étant pertinent.

La cinquième question, « Avez-vous des commentaires sur la mise en forme ou l'aspect esthétique de la grille ? », a permis la collection de trois réponses. La première est de retirer les deux premières colonnes de la grille, soit les entrées du questionnement didactique et les éléments considérés pour ces entrées, pour ne garder que les optiques et les attributs. Nous considérons que cette suggestion permettra de simplifier le premier coup d'œil jeté à la grille et nous décidons donc d'utiliser cette suggestion dans la version améliorée de la grille. La deuxième réponse est d'ajouter une colonne avec un langage de programmation utilisé comme exemple. La version initiale de la grille comportait à titre indicatif deux colonnes vides pour les langages de programmation nº1 et nº2. Nous décidons de retenir cette suggestion et de remplir la colonne pour le langage nº1 à l'aide d'un langage de programmation auquel s'intéresse le chercheur puisqu'il l'utilise dans le cours introductif qu'il enseigne : le langage de programmation Typescript. Cette colonne exemple contribuera à renforcer la guidance que la grille fournit à ses utilisatrices et utilisateurs. La dernière réponse propose l'ajout de lignes dans la grille pour démontrer des exemples de programmes complets et puisque cela augmenterait la complexité de la grille davantage, nous décidons de ne pas retenir cette suggestion.

À la sixième question, « Avez-vous des commentaires sur le format de fichier utilisé pour vous transmettre la grille ? », tous les participants ont répondu qu'ils n'avaient aucun commentaire à formuler. Ainsi, cette question ne pourra pas être considérée pour l'amélioration de la grille.

Pour la septième et dernière question, deux participants proposent de diviser la grille en deux si une réduction de sa longueur est impossible. Deux façons de diviser la grille ont été proposées : soit par dichotomie entre les attributs en lien avec le langage de programmation et les attributs en lien avec l'environnement de développement ou par dichotomie entre les attributs de base et les attributs avancés. Puisqu'il est possible de réduire la taille de la grille à l'aide des commentaires émis par les pairs et que nous considérons important que la grille demeure un objet atomique permettant une analyse en profondeur des impacts didactiques, nous optons pour la stratégie de réduction de la taille plutôt que de diviser la grille en deux parties.

La liste suivante résume les modifications qui seront apportées à la grille suite à l'analyse des données collectées :

1. Ajouter des instructions d'utilisation dans l'entête de la grille ;
2. Réduire la complexité de la grille en retirant les deux premières colonnes de la grille ;
3. Réduire la taille de la grille à l'aide des modifications suivantes :
    a. Retirer les attributs en lien avec l'élément 4.2 de la grille, soit la taxonomie des systèmes pour programmeurs débutants de Kelleher et Pausch (2005) ;
    b. Retirer tous les attributs trop génériques ne permettant pas de discriminer entre les langages les plus communs :
        i. Support pour les séquences ;
        ii. Support pour les alternatives ;
        iii. Support des répétitives ;
        iv. Support pour les procédures ou fonctions sans valeur de retour ;
        v. Support pour les fonctions ;
        vi. Support pour tableaux unidimensionnels;
        vii. Éditeur : support pour copier-coller ;

c. Retirer l’attribut « Indices de tableau débutant à zéro, un ou au choix » ;

4. Remplir la colonne « Langage de programmation nº1 » en utilisant le langage Typescript pour donner un exemple d’utilisation de la grille ;
5. Retirer les attributs « Aucun » ou faisant référence à un autre élément de la grille puisqu’ils sont inutiles.

5. LA VERSION AMÉLIORÉE DE LA GRILLE

Les modifications apportées à la grille d’analyse des langages de programmation ont permis de réduire le nombre d’attributs de 128 à 73. De plus, deux colonnes ont été retirées et une colonne vide a été remplie avec les données d’un langage de programmation exemple. Avec ces modifications, l’outil est plus apte à satisfaire l’objectif général de la recherche, soit de permettre à un enseignant, une enseignante ou une équipe de sélectionner le langage de programmation le plus approprié pour leurs étudiants et étudiantes, et pour faciliter leur rapport aux savoirs algorithmiques.

Voici la nouvelle grille :

Tableau 7 Version améliorée de la grille d'analyse des langages de programmation

Grille d'analyse des langages de programmation (version 2)

par Stéphane Duguay (stephane.duguay@cegep-rimouski.qc.ca)

Instructions : La grille est une grille d'analyse et non une grille d'évaluation. Il n'y a donc pas de pondération associée aux attributs et aucun pointage final ne sera généré indiquant qu'un langage est plus approprié qu'un autre. La diversité des contextes d'application, des considérations, des contraintes et des valeurs guidant la sélection d'un langage de programmation prohibe une approche trop rigide. Il faut donc voir cette grille comme un guide d'analyse vous permettant d'avoir une réflexion didactique sur un ou des langages de programmation. De plus, il n'existe pas de bonne ou de mauvaise façon d'utiliser la grille ou de la remplir, tant que votre démarche vous semble utile.

Optiques considérées	**Attributs**	**Exemple : Typescript**	**Votre langage de programmation**
La structure générale d'un programme	Fonction principale (point d'entrée unique)	Non, mais une fonction main() peut être créée et appelée par un environnement éducatif.	

	Point(s) de sortie unique ou multiple	Points de sortie multiples possibles.	
	La non-utilisation des variables globales	Optionnellement	
Les variables	Syntaxe de déclaration des variables	let nomVariable : type = valeur;	
	Contraintes relatives au nommage des variables	Habituelles : noms de variables débutant par une lettre ou le caractère souligné.	
	Typage statique (types acquis à la compilation) versus Typage dynamique (types acquis à l'exécution)	Typage statique encouragé, mais supporte les deux.	
	Fortement typé (peu de conversions ou transtypages implicites)	Fortement typé lorsque la variable	

	versus Faiblement typé (beaucoup de conversions et transtypages implicites)	est statiquement typée.	
Les paramètres	Support pour paramètres par valeur	Oui	
	Support pour paramètres par référence	Seulement les objets	
Les variables tableaux	Support pour tableaux multidimensionnels	Oui	
	Contraintes associées au passage des tableaux en paramètre	Par référence seulement	
Écriture et modification du code selon (Gouvernement du Québec, 2013)	Éditeur intégré à l'environnement de développement	Visual Studio : oui	
	Éditeur : support pour la coloration syntaxique	Visual Studio : oui	

	Éditeur : support pour le complètement automatique	Visual Studio : oui	
Débogage du code selon (Gouvernement du Québec, 2015)	Débogueur intégré à l'environnement de développement	Visual Studio : oui, requiert l'utilisation d'un navigateur Microsoft	
	Débogueur : point d'arrêt sur instruction	Visual Studio : oui	
	Débogueur : l'inspection des variables	Visual Studio : oui	
	Débogueur : exécution contrôlée supportant le pas-à-pas et le pas-à-pas approfondi	Visual Studio : oui	
Programmation en équipe	Environnement de développement intégré supportant la mise en commun de code source	Requiert des outils externes	

	Environnement de développement intégré supportant la communication publique et privée entre les étudiants et étudiantes, et avec l'enseignant	Requiert des outils externes	
Programmation en binôme ou en aquarium	Éditeur : support pour l'affichage des numéros de ligne	Visual Studio : oui	
La préparation de l'environnement de programmation	Existence de raccourcis clavier	Visual Studio : oui	
	Support pour l'utilisation des mnémoniques clavier	Visual Studio : oui	
	Support pour la personnalisation des raccourcis clavier	Visual Studio : oui	
	Support pour la personnalisation de la	Visual Studio : oui	

	police de caractère utilisée dans l'éditeur		
	Support pour la personnalisation des couleurs du texte et du fond d'écran dans l'éditeur	Visual Studio : oui	
	Surbrillance de la syntaxe adaptée au langage de programmation	Visual Studio : oui	
L’adaptation de l’algorithme aux contraintes du langage de programmation	Système de types de données le plus naturel possible	Oui : 3 types de base (string, number et boolean).	
	Opérations d'entrées et sorties simplifiées	Non : web ou console.log	
	Support pour toutes les structures de contrôle utilisées en algorithmique	Oui	

	Syntaxe confortable se rapprochant du langage naturel	Moyen	
	Règles sémantiques et paradigmes appropriés pour un cours introductif.	Oui, permet la programmation structurée sans introduction d'éléments de programmation orientée-objet.	
La compilation du programme	Compilation automatique lors de l'exécution	Visual Studio : oui	
	Repérage facile des erreurs de compilation	Visual Studio : oui	
	Avertissements pertinents ou configurables	Visual Studio : oui	
La validation du programme	Support intégré pour les tests unitaires	Visual Studio : oui	

Langue maternelle de la programmeuse ou du programmeur	Langue utilisée pour les mots-clés du langage de programmation	Anglais	
Patrimoine intellectuel en mathématiques de la programmeuse ou du programmeur	Opérateurs mathématiques familiers	Opérateurs habituels	
Erreurs dont l'origine dépend de la compréhension qu'ils ont du modèle d'exécution d'un programme	Historique des valeurs de variables	Visual Studio : non	
	Surbrillance des variables lors du changement de valeur	Visual Studio : oui	
	Pile des appels facile d'approche	Visual Studio : non	
	Valeurs de retour des fonctions présentées dans l'outil d'inspection des variables	Visual Studio : oui	
La connaissance qu'a l'enseignante ou enseignant du langage de programmation	Niveau de compétence de l'enseignante ou enseignant dans ce langage	Dépendant de l'enseignant	

Production de notes de cours	Langage facilement incorporable dans un document Word, OpenOffice ou LaTeX	Non supporté par les paquets LaTeX pour l'intégration de code source.	
Création de programmes exemples	Code des exemples est autosuffisant (isolé, sans dépendances)	Non, HTML nécessaire. Peut être standardisé par l'utilisation d'un environnement éducatif.	
	Support pour les commentaires	Oui	
	Support pour les commentaires multilignes	Oui	
Création d'exercices	Environnement de développement permet le contrôle des entrées à l'aide de fichiers de donnée pour permettre l'autocorrection.	Non	

Création d'évaluations	Environnement de développement permet le contrôle des entrées et la validation des sorties à l'aide de fichiers de donnée pour permettre la correction automatisée.	Non	
Si environnement web, le site web est accessible avec les principaux navigateurs modernes qui respectent les standards établis	HTML et CSS	Oui	
	N'utilisant pas les technologies propriétaires	Aucune, sauf pour le débogueur dans Visual Studio.	
Fourni dans un standard reconnu lisible sur au moins l'un des principaux dispositifs et systèmes d'exploitation actuels	Disponible Windows, Linux ou MacOS	Oui	
	Disponible sur PC ou Mac	Oui	
	Disponible en version web	Non	
Ne présente aucune erreur structurelle	Aucun bogue flagrant	Visual Studio : aucun	
	Les menus provoquent tous une opération	Visual Studio : oui	

	Aucune erreur structurelle dans la documentation	Bien documenté	
Donne des indications techniques adaptées à la clientèle concernant l'utilisation et le fonctionnement	Messages d'erreur courts	Visual Studio : acceptable	
	Messages d'erreur compréhensibles	Visual Studio : oui	
	Système de complétion automatique pour le code source	Visual Studio : oui	
Donne accès à une fonction "Aide", dans le matériel et en ligne, en cas de problème	Aide contextuelle disponible	Visual Studio : oui	
	Index des rubriques d'aide	Visual Studio : oui	
	Outil de recherche dans les rubriques d'aide	Visual Studio : oui	
	Aide disponible en ligne	Visual Studio : oui	
	Aide disponible hors ligne (sans connexion internet)	Visual Studio : oui	
L’interface propose des éléments facilitant la lisibilité du matériel affiché à l’écran	Choix de la police de caractère et de sa taille	Visual Studio : oui	

	Choix des couleurs	Visual Studio : oui	
	Permet le zoom	Visual Studio : oui	
La navigation est intuitive	Interface avec prise en main facile	Visual Studio : oui	
	Interface réutilisant des paradigmes établis	Visual Studio : oui	
	Appropriation facile par l'utilisateur des opérations du logiciel	Visual Studio : oui	
Vision béhavioriste de l'apprentissage	Facilitant les exercices pratiques et les examens théoriques	Non, doit concevoir un environnement éducatif dans lequel les exercices seront exécutés.	
Vision cognitiviste de l'apprentissage	Facilitant les projets	Oui, disponibilité de librairies et de moteurs de jeux (compatibilité avec Javascript).	

CONCLUSION

Cet essai a décrit la recherche effectuée lors de la conception d'une grille d'analyse des langages de programmation pour l'apprentissage et l'enseignement de l'algorithmique dans le cadre de cours introductifs à la programmation du programme de formation de Techniques de l'informatique au collégial québécois. Appuyés sur un cadre de référence se basant principalement sur le questionnement didactique selon Lapierre (2008) et à l'aide d'une méthodologie de recherche développement proposée par Loiselle et Harvey (2009), nous avons créé une grille d'analyse et considérons avoir généralement atteint l'objectif général de la recherche. Les chapitres sur la problématique, le cadre de référence, la méthodologie ainsi que la présentation et l'interprétation des résultats sont les rendus de la démarche entreprise. Par contre, la grille, soit dans sa version initiale ou améliorée, pourrait être revalidée auprès d'un échantillon plus grand et peut-être que cela permettrait la création d'une meilleure version de la grille.

Le produit de cette recherche contribuera à améliorer le constat énoncé dans le chapitre couvrant la problématique : en effet, les responsables du choix d'un langage de programmation pour les cours introductifs ont maintenant un outil pour travailler. Par contre, il serait naïf de prétendre que la problématique identifiée est complètement réglée par le seul fait de la création de la grille d'analyse des langages de programmation… encore faut-il que la grille soit utilisée. Le chercheur principal s'engage à tenter de diffuser le produit de cette recherche dans le réseau collégial.

En raison du fait que le recrutement de personnes participantes fut difficile et que la moitié de l'échantillon se soit retirée de la recherche avant la collecte des données, on pourrait croire que les enseignants et enseignantes du réseau collégial n'ont pas l'intérêt ou n'ont tout simplement pas le temps de participer au développement d'outils et de méthodes spécifiques à l'apprentissage et l'enseignement de la

programmation. Il n’est donc pas certain qu’ils ou elles auront l’intérêt ou le temps pour utiliser une grille, même si celle-ci est diffusée.

La didactique de l’algorithmique et de la programmation est plutôt jeune dans le domaine de l’éducation au Québec et dans les communautés francophones. Nous devons travailler à développer un cadre de référence solide pour l’apprentissage et l’enseignement de notre discipline. Pour d'éventuels chercheurs ou chercheuses intéressés à poursuivre ce travail, voici quelques pistes possibles. Une piste de développement majeure et nécessitant beaucoup de ressources serait de mesurer quantitativement les effets des attributs d’un langage sur la réussite des étudiants et étudiantes. De plus, une simple veille didactique pour des grilles d’analyse des langages de programmation au niveau international pourrait aussi être efficace, mais celles-ci nécessiteront sûrement une adaptation au contexte collégial québécois.

RÉFÉRENCES BIBLIOGRAPHIQUES

Bard, P. (1991). *L'algorithmique pas à pas à l'aide du langage Pascal*. Laval: Éditions Beauchemin.

Benabbou, F., et Hanoune, M. (2006). EasyAlgo: un environnement d'apprentissage et d'autoévaluation de l'algorithmique. *Automates intelligents, 76*, 1-15. Document téléaccessible à l'adresse <http://admiroutes.asso.fr/larevue/2006/76/Article_Apprentissage.pdf>.

Cormen, T. H. (2013). *Algorithmes: notions de base*. Paris: Dunod.

Cégep de Rimouski (2016). *Plan du cours 420-131-RK*. Rimouski: Cégep de Rimouski, département d'informatique.

Fortin, M.-F. (2010). *Fondements et étapes du processus de recherche* (2e éd.). Montréal: Chenelière Éducation (1re éd. 2006).

Gouvernement du Québec (2000). *Administration, commerce et informatique. Techniques informatique. Programme d'études 420.A0*. Québec: Ministère de l'Éducation.

Gouvernement du Québec (2010). *L'approbation du matériel didactique*. Québec: Ministère de l'Éducation, du Loisir et du Sport, Direction des ressources didactiques. Document téléaccessible à l'adresse <http://www1.education.gouv.qc.ca/bamd/Doc/Approbation_materiel_didactique_fr.pdf>.

Gouvernement du Québec (2013). *Étude sectorielle sur les besoins de main-d'œuvre en TIC : rapport de recherche*. Québec: Ministère de l'Éducation, du Loisir et du Sport.

Gouvernement du Québec (2015). *Technicienne et technicien en informatique: rapport d'analyse de profession*. Québec: Ministère de l'Éducation, de l'Enseignement supérieur et de la Recherche, Direction des programmes de formation technique.

Gouvernement du Québec (2016). *Cadre de référence pour l'évaluation du matériel didactique: cadre numérique*. Québec: Ministère de l'Éducation et de l'Enseignement supérieur, Direction des ressources didactiques. Document téléaccessible à l'adresse <http://www1.education.gouv.qc.ca/bamd/Doc /Cadre_numerique.pdf>

Guibert, N., Guittet, L. et Girard, P. (2005). *A study of the efficiency of an alternative programming paradigm to teach the basics of programming.* Communication présentée au *8th World Conference on Computers in Education* de l'IFIP, Cape Town, Afrique du Sud, 4 au 7 juillet.

Johansen Z., Auer K., Button B., Cockburn A., Grenning J., Johansen K. et O'Melia D. (2002). Extreme Fishbowl. *In* D. Wells et L. A. William (dir.), *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods* (p. 289). Londres: Springer-Verlag.

Karsenti, T. et Savoie-Zajc, L. (2011). *La recherche en éducation : étapes et approches* (3e éd.). St-Laurent: Les Éditions du Renouveau Pédagogique (1re éd. 2004).

Kelleher, C. et Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computer Survey*, *37*(2), 83-137.

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory into practice*, *41*(4), 212-218.

Lapierre, L. (2008). Un cadre de référence pour le questionnement didactique au collégial. *Pédagogie collégiale (AQPC)*, *21*(2), 5-12.

Loiselle, J. (2001). La recherche développement en éducation: sa nature et ses caractéristiques. *Nouvelles dynamiques de recherche en éducation*, 77-97.

Loiselle, J., & Harvey, S. (2007). La recherche développement en éducation: fondements, apports et limites. *Recherches qualitatives*, *27*(1), 40-59.

Loiselle, J. et Harvey, S. (2009). Proposition d'un modèle de recherche développement. *Recherches qualitatives*, *28*(2), 95-117.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. *ACM SIGCSE Bulletin*, *34*(1), 38-42.

Ménard, L. et St-Pierre L. (2014). *Se former à la pédagogie de l'enseignement supérieur*. Montréal : Association québécoise de pédagogie collégiale (AQPC).

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). Improving the CS1 experience with pair programming. *ACM SIGCSE Bulletin*, *35*(1), 359‑362.

Office québécois de la langue française (s.d.). Fiche terminologique du mot « algorithmique ». *Grand dictionnaire terminologique en ligne*. Site téléaccessible à l'adresse <http://www.granddictionnaire.gouv.qc.ca /ficheOqlf.aspx?Id_Fiche=8361077>. Consulté le 26 avril 2018.

PERFORMA (2017). *Guide de présentation du bloc recherche innovation et analyse critique de la maîtrise en enseignement au collégial*. Sherbrooke: Université de Sherbrooke, Faculté d'éducation, secteur PERFORMA.

Prégent, R. (1990). *La préparation d'un cours*. Montréal: Éditions de l'École polytechnique.

Prud'homme, A. (2015). Apprendre de ses expériences professionnelles grâce à une démarche de résolution de problèmes. *Pédagogie collégiale (AQPC)*, *28*(4), 38–44.

Massachusetts Institute of Technology (s.d.). *À propos de Scratch*. Massachusetts Institute of Technology, MIT Media Lab, Lifelong Kindergarten Group. Site téléaccessible à l'adresse <https://scratch.mit.edu/about/>. Consulté le 11 avril 2016.

Vergnaud, G. (1999). A quoi sert la didactique. *Sciences Humaines, la dynamique des savoirs, 24*.

Williams, L., & Upchurch, R. L. (2001). In support of student pair-programming. *ACM SIGCSE Bulletin*, *33*(1), 327‑331.

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE software*, *17*(4), 19-25.

Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. *ACM Inroads*, *2*(4), 71‑76.

ANNEXE A

La compétence 016W du devis ministériel pour Techniques de l'informatique (Gouvernement du Québec, 2000) : « Produire des algorithmes »

Tableau 8 Énoncé de la compétence « Produire des algorithmes »

OBJECTIF	STANDARD
Énoncé de la compétence	**Contexte de réalisation**
Produire des algorithmes.	• À partir de situations variées représentatives du milieu de travail. • À partir d'une station de travail et des logiciels appropriés. • À partir des exigences de l'entreprise. • À l'aide des manuels de référence techniques appropriés.
Éléments de la compétence	**Critères de performance**
1 Analyser la situation.	1.1 Établissement correcte des données d'entrée. 1.2 Établissement correcte des données de sortie. 1.3 Établissement correcte de la nature des traitements. 1.4 Détermination correcte des conditions d'exécution de l'algorithme.
2 Mettre au point l'algorithme.	2.1 Choix d'un mode de représentation de l'algorithme conforme aux exigences de l'entreprise. 2.2 Détermination d'une séquence logique des opérations. 2.3 Détermination des structures de traitement appropriées à chacune des opérations. 2.4 Application rigoureuse des règles de syntaxe propres au mode de représentation retenu. 2.5 Recherche d'une solution algorithmique efficace. 2.6 Représentation précise de la solution algorithmique retenue. 2.7 Présence de toute l'information nécessaire à l'interprétation de l'algorithme.
3 Valider l'algorithme.	3.1 Vérification de la pertinence de la solution compte tenu de la situation initiale. 3.2 Détermination des erreurs et des lacunes de la solution algorithmique mise au point. 3.3 Modification appropriée de la solution algorithmique.

ANNEXE B

La compétence 016S du devis ministériel pour
Techniques de l'informatique (Gouvernement du Québec, 2000) :
« Exploiter un langage de programmation structurée »

Tableau 9 Énoncé de la compétence « Exploiter un langage de programmation structurée »

OBJECTIF	STANDARD
Énoncé de la compétence	**Contexte de réalisation**
Exploiter un langage de programmation structurée.	• À partir d'une station de travail et des logiciels appropriés. • À partir d'algorithmes valides et représentatifs du milieu de travail. • À partir des normes et des exigences de l'entreprise. • À l'aide des manuels de référence techniques appropriés à l'environnement de programmation.
Éléments de la compétence	**Critères de performance**
1 Préparer l'environnement de programmation.	1.1 Vérification méthodique de l'accès aux différents éléments physiques et logiques de l'environnement. 1.2 Configuration de l'environnement appropriée aux caractéristiques de la situation. 1.3 Personnalisation de l'environnement efficace et conforme aux exigences de l'entreprise.
2 Adapter l'algorithme aux contraintes du langage de programmation.	2.1 Modification appropriée de la représentation des données. 2.2 Adaptation correcte des conditions d'exécution. 2.3 Modification appropriée des structures de traitement. 2.4 Adaptation appropriée de la séquence des opérations.

3	Traduire l’algorithme dans le langage de programmation.	3.1 Utilisation efficace des fonctionnalités d’édition de l’environnement.
		3.2 Application des règles de syntaxe et de sémantique propres au langage utilisé.
		3.3 Application rigoureuse des standards de codification.
		3.4 Application judicieuse des principes de la programmation structurée.
		3.5 Mise à profit judicieuse des possibilités du langage.
		3.6 Consignation des commentaires pertinents et conformes aux exigences de l’entreprise.
4	Compiler le programme.	4.1 Utilisation efficace des fonctionnalités de compilation de l’environnement.
		4.2 Repérage des erreurs de compilation.
		4.3 Correction des erreurs de compilation.
5	Valider le programme.	5.1 Utilisation efficace des fonctionnalités d’exécution et de débogage de l’environnement.
		5.2 Préparation correcte des jeux d’essai nécessaires à la vérification du fonctionnement du programme.
		5.3 Interprétation juste des résultats.
		5.4 Débogage approprié du programme selon l’algorithme.

ANNEXE C

Plan pour l'entrevue semi-dirigée

QUESTIONNAIRE POUR L'ENTREVUE SEMI-DIRIGÉE
PAR VIDÉOCONFÉRENCE

Date et heure de l'entrevue :

_____

Nom du participant :

_____

Identificateur pour vidéoconférence :

_____

Bonjour,

Je vous remercie de prendre de votre temps pour répondre à mes questions.

1- Sur quel(s) langage(s) de programmation avez-vous appliqué la grille d'analyse ?
2- Avez-vous l'impression que la grille a couvert tous les aspects du langage en lien avec son utilisation dans votre contexte éducatif ?
    - Sinon, quel(s) manque(s) avez-vous identifié(s) ?
3- Quels attributs de la grille vous ont semblé les plus pertinents ?
    - Pourquoi ?
4- Quels attributs de la grille vous ont semblé les moins pertinents ?
    - Pourquoi ?
5- Avez-vous des commentaires sur la mise en forme ou l'aspect esthétique de la grille ?
6- Avez-vous des commentaires sur le format de fichier utilisé pour vous transmettre la grille ?
7- Y a-t-il des sujets qui n'ont pas été abordés et sur lesquels vous voudriez ajouter quelque chose ?

Je vous remercie de votre participation.

Stéphane Duguay

ANNEXE D

Grille de classification des réponses collectées

Tableau 10 Grille de classification des réponses collectées

Question	**Réponses** Pour chaque réponse : le nombre de participants l'ayant donnée suivi de la liste des personnes participantes.
1. Sur quel(s) langage(s) de programmation avez-vous appliqué la grille d'analyse ?	Java (1: nº2), C++ (1: nº2) et Python (1: nº4)
2. Quel(s) manque(s) avez-vous identifié(s) dans la grille ?	• Pas d'instructions d'utilisation (3: nº1, nº2, nº4) • Trop longue ou complexe (4: nº1, nº2, nº3, nº4)
3. Quels attributs de la grille vous ont semblé les plus pertinents ?	• Attributs en lien avec les modules (1: nº2) • Attributs en lien avec les types de paramètres (1: nº2) • Attributs en lien avec le débogueur (1: nº4) • Attribut « langue utilisée pour les mots-clés du langage de programmation » (1: nº4) • Attribut « niveau de compétence de l'enseignante ou enseignant dans ce langage » (1: nº4)
4. Quels attributs de la grille vous ont semblé les moins pertinents ?	• Attributs en lien avec les types de paramètres (2: nº3, nº4) • Attributs tirés de la taxonomie des systèmes pour programmeurs débutants (Kelleher et Pausch, 2005) (1 : nº2) • Tout ce qui est trop générique et répandu (1: nº4) • Attribut « Indices de tableau débutant à zéro, un ou au choix » (1: nº4) • Tous les attributs en lien avec le matériel didactique (1: nº4)
5. Avez-vous des commentaires sur la mise en forme ou l'aspect esthétique de la grille ?	• Les deux premières colonnes sont inutiles (1: nº4) • Ajouter une colonne avec un langage de programmation exemple (1: nº4) • Ajouter du code de programmes simples classiques pour permettre une comparaison holistique (1: nº4)
6. Avez-vous des commentaires sur le format de fichier utilisé pour vous transmettre la grille ?	• Aucun
7. Y a-t-il des sujets qui n'ont pas été abordés et sur lesquels vous voudriez ajouter quelque chose ?	• Séparer la grille en deux si réduction de taille impossible (2 : nº2, nº4)

ANNEXE E

Lettre d'information et formulaire de consentement

LETTRE D'INFORMATION ET FORMULAIRE DE CONSENTEMENT

Titre du projet de recherche : Conception d'une grille d'analyse des langages de programmation utilisés en algorithmique dans le programme Techniques de l'informatique.

Chercheur : Stéphane Duguay, enseignant en informatique, Cégep de Rimouski
Programme de maîtrise en enseignement au collégial
PERFORMA, Faculté d'éducation, Université de Sherbrooke

Directeur d'essai : Christian Barrette, chargé de cours PERFORMA, Université de Sherbrooke

Collègues enseignantes et enseignants en informatique,

Vous enseignez des cours de programmation ? Comment avez-vous choisi le langage de programmation que vous utilisez pour initier vos étudiants à la programmation ? Ce langage est-il conçu pour des professionnels de l'informatique et cela cause-t-il des difficultés dans son utilisation pédagogique ? Nous sommes plusieurs dans cette situation !

Je suis Stéphane Duguay, enseignant au cégep de Rimouski. Dans le cadre d'une maitrise en enseignement au collégial au secteur PERFORMA de l'Université de Sherbrooke, ma recherche de type « développement » s'intéresse principalement aux attributs des langages de programmation dans une optique d'enseignement et d'apprentissage. Elle a pour objectif général de concevoir une grille d'analyse des langages de programmation pour l'apprentissage et l'enseignement de l'algorithmique dans le cadre du programme Techniques de l'informatique au collégial québécois pour permettre à un enseignant, une enseignante ou une équipe d'enseignants de sélectionner le langage de programmation le plus approprié pour les étudiants et étudiantes et pour faciliter leur rapport au savoir algorithmique. Pour atteindre cet objectif, nous allons dégager les attributs d'un langage de programmation pour l'apprentissage et l'enseignement de l'algorithmique adaptés aux besoins du niveau collégial québécois, mettre au point une grille d'analyse des langages de programmation à partir des attributs dégagés et valider la grille d'analyse auprès de pairs.

Je suis donc à la recherche d'enseignantes et d'enseignants volontaires pour valider la grille d'analyse. Votre rôle en tant que participante ou participant à la recherche consistera en :

1. Appliquer la grille d'analyse sur un langage de programmation de votre choix (environ 20 minutes) ;
2. Donner votre avis et faire part de suggestions sur la grille lors d'une entrevue semi-dirigée par téléphone ou vidéoconférence (environ 45 minutes).

Votre aide permettra d'améliorer la grille afin qu'elle devienne un outil efficace et personnalisable pour choisir un langage de programmation adapté aux différents contextes de l'enseignement de l'informatique au collégial.

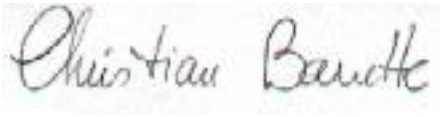
Pour éviter votre identification comme personne participante à cette recherche, les données recueillies seront traitées de manière entièrement confidentielle et ne pourront en aucun cas mener à votre identification. Les données recueillies auprès de vous ne serviront qu'au chercheur. Cette confidentialité sera assurée par l'anonymisation des données recueillies pour la publication de l'essai ainsi qu'une mise sous clé numérique (encodage) des données originales dont seul le chercheur aura connaissance. Ces données originales seront détruites un an après l'acceptation de l'essai par l'Université de Sherbrooke et ne serviront seulement qu'à mener à terme cette recherche et l'essai l'accompagnant.

Votre participation ne sera pas rémunérée et se fait sur une base volontaire. Vous êtes entièrement libre de participer ou non à cette recherche et vous êtes aussi libre de vous retirer en tout temps, sans préjudice. Les risques associés à votre participation sont quasi inexistants et je m'engage à mettre en œuvre les moyens nécessaires pour pallier à d'éventuelles conséquences négatives non prévues.

Votre contribution à l'avancement de la recherche sur l'utilisation des langages de programmation dans un contexte éducatif ainsi qu'un accès privilégié à la première version de la grille sont les bénéfices tangibles de votre participation à cette recherche.

Si vous voulez plus de détails ou pour obtenir des réponses à vos questions, contactez-moi ou contactez mon directeur d'essai à l'aide des informations suivantes :

Stéphane Duguay
Enseignant en informatique, Cégep de Rimouski
stephane.duguay@cegep-rimouski.qc.ca
418-723-1880 poste 2104



Christian Barrette

Chargé de cours, PERFORMA, Université de Sherbrooke

christian.barrette@usherbrooke.ca

450-812-3714

J'ai lu et compris la lettre d'information au sujet du projet de recherche. J'ai compris les conditions, les risques et les bénéfices de ma participation. J'ai obtenu des réponses aux questions que je me posais au sujet de ce projet. J'accepte librement de participer à ce projet de recherche, sans rémunération.

Participante ou participant

_____

Signature

_____

Nom

_____

Date

Chercheur

_____

Signature

Stéphane Duguay_____

Nom

9 octobre 2017_____

Date